UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS – DCEN
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALISSON DE OLIVEIRA MAURÍCIO

SISTEMA WEB PARA A GESTÃO DE CLIENTES DA ECOS SISTEMAS

MOSSORÓ-RN
2014

ALISSON DE OLIVEIRA MAURÍCIO

SISTEMA WEB PARA A GESTÃO DE CLIENTES DA ECOS SISTEMAS

Relatório apresentado à Universidade Federal Rural do Semi-Árido – UFERSA, Departamento de Ciências Exatas e Naturais, para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profª. MSc. Flávia Estélia Silva Coelho.

MOSSORÓ-RN

2014

O conteúdo desta obra é de inteira responsabilidade de seus autores

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência

M454s Mauricio, Álisson de Oliveira.

Sistema web para a gestão de clientes da ecos sistemas / Álisson de Oliveira Mauricio. -- Mossoró, 2014.

48f.: il.
Orientadora: Profª. M. Sc. Flávia Estélia Silva Coelho.

Monografia (Graduação em Ciência da Computação) – Universidade Federal Rural do Semi-Árido. Pró-Reitoria de Graduação.

1. Sistema web. 2. Framework bootstrap. 3. Framework grails. 4. Gestão de clientes. 5. Linguagem groovy. 6. Metodologia scrum. I. Título.

RN/UFERSA/BCOT /644-14 CDD: 004.678

Bibliotecária: Vanessa de Oliveira Pessoa
CRB-15/453

ALISSON DE OLIVEIRA MAURÍCIO

SISTEMA WEB PARA A GESTÃO DE CLIENTES DA ECOS SISTEMAS

Parecer dos Professores:

.....

.....

.....

Data da Defesa: 04/08/2014

Profa. MSc. Flávia Estélia Silva Coelho – UFERSA
Presidente

Profa. DSc. Angélica Félix de Castro – UFERSA
Primeiro Membro

Profa. DSc. Danielle Simone da Silva Casillo – UFERSA
Segundo Membro

À minha família e aos meus amigos, que são a base e a pedra angular da minha vida.

AGRADECIMENTOS

A Deus que me deu o dom da vida e do pensar e pelas coisas boas que fez em minha vida.

Aos meus pais, Francisco Lindeci e Lucina Moreira, que batalharam para que eu chegasse aonde eu cheguei e pelos ensinamentos que irei guardar por toda a vida. O apoio e a determinação se fez constante nos maus e bons momentos da minha vida.

Às minhas irmãs, Aline e Alice, que me estenderam as mãos em todos os momentos da minha vida. O riso, um dos elementos essenciais à vida, é garantido quando estou com elas.

Aos meus amigos para a vida toda, obrigado pelo apoio e por fazer da minha vida uma festa que não tem fim. Sei que posso contar com vocês e vocês podem contar comigo.

À minha caríssima orientadora, Profª. Flávia Coelho, muito obrigado pela orientação e paciência dedicadas a mim. Saiba que a senhora faz jus ao título de Professora, pois tem uma didática ímpar.

Aos meus colegas de faculdade, obrigado por me acompanharem nessa jornada e pelos conselhos que me deram nos momentos em que pensei que não dava mais e queria desistir. Sem vocês, eu não teria concluído a minha graduação.

Às professoras que fizeram parte da minha banca, Profª. Danielle Casillo e Profª. Angélica Castro, obrigado por terem contribuído com este trabalho ao longo da sua realização.

Aos professores da minha graduação que, além do saber técnico, me educaram para a vida.

Aos meus tios, tias, primos e primas, que, de maneira singular, contribuíram para que eu fosse até o fim, apesar das adversidades da vida.

E por último, mas não menos importante, agradeço à empresa ECOS Sistemas pela oportunidade de estágio e pela aquisição de conhecimentos profissionais. Local onde eu aprendi muito e conheci pessoas maravilhosas.

“Não conheço nenhuma fórmula infalível para obter o sucesso, mas conheço uma forma infalível de fracassar: tentar agradar a todos.”

(John F. Kennedy)

RESUMO

Uma das atividades mais importantes nas empresas que desenvolvem *software* é a gestão de clientes. Dependendo do número de clientes e do sistema utilizado para gerenciá-los, a tarefa pode ser árdua. O presente trabalho propõe o *Ecos Manager Internal* (EMI), que é uma solução sob a forma de um sistema WEB para o problema do gerenciamento dos clientes da empresa ECOS Sistemas. O método utilizado para o desenvolvimento foi o *Scrum*, que consiste em desenvolver sistemas de forma rápida e com garantia de qualidade. A linguagem de programação utilizada para o desenvolvimento de sistema foi o *Groovy*, que é considerada uma versão simplificada de *Java*. A alta-produtividade foi garantida ao utilizar os *frameworks Grails* e *Bootstrap*. O ambiente de desenvolvimento utilizado foi o *Groovy/Grails Tool Suite*, que é próprio para o desenvolvimento de sistemas que utilizam o *Grails* e o *Groovy*. O EMI solucionou o problema do sistema legado que a empresa utilizava para gerenciar os seus clientes, ao migrar para uma tecnologia mais recente, e unificou os bancos de dados da empresa, com os quais ela trabalhava.

Palavras-chave: Gestão de clientes. Aplicação para a Web. *Framework Grails*. *Linguagem Groovy*. *Framework Bootstrap*. Metodologia *Scrum.*

LISTA DE FIGURAS

LISTA DE LISTAGENS

LISTA DE TABELAS

LISTA DE SIGLAS

CNPJ	Cadastro Nacional de Pessoa Jurídica
CPF	Cadastro de Pessoas Físicas
CSS	*Cascading Style Sheets*
ECOS	Empresa Comercial de Sistemas LTDA
EMI	*Ecos Manager Internal*
GGTS	*Groovy/Grails Tool Suite*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JCP	*Java Community Process*
JSR	*Java Specification Request*
MVC	*Model-View-Controller*
PDF	*Portable Document Format*
RN	Rio Grande do Norte
SEC	Sistema ECOS de Contabilidade
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WEB	*World Wide Web*

SUMÁRIO

1. INTRODUÇÃO

A ECOS Sistemas, empresa de desenvolvimento de sistemas para a área contábil, surgiu no início da década de 90 quando a informática dava os seus primeiros passos na utilização de gestores eletrônicos para rotinas profissionais e empresariais.

No ano de 1991, o contador José Alves Neto (*in memoriam*) obteve sucesso ao unir a sua experiência em rotinas contábeis com a inovação da era da informatização, criando assim, o Sistema ECOS de Contabilidade (SEC). Com isso, tornou possível à classe contábil do RN, o acesso aos recursos de aplicativos que antes eram importados do centro-sul do país.

Com pouco mais de 20 anos no mercado de trabalho, a empresa ECOS já se consolidou como uma excelente empresa no desenvolvimento de sistemas contábeis e conta com um pouco mais de 550 clientes distribuídos em quase todos os estados da região nordeste do Brasil.

No começo, o trabalho de gerenciamento dos seus clientes era simples, em função do número de clientes. Com o tempo, a quantidade foi crescendo e surgiu a necessidade da criação de um *software desktop* para gerenciá-los. Hoje, a quantidade de clientes já é bem maior e o sistema de gerenciamento se tornou um sistema legado, pois a linguagem de programação na qual o software foi desenvolvido já se tornou obsoleta e não foi efetuada a sua migração para uma linguagem mais atual. Portanto, atualmente, o gerenciamento de clientes da empresa ECOS Sistemas é um trabalho árduo.

O objeto de estudo deste trabalho consiste em propor, descrever e desenvolver uma solução para o problema do uso de um sistema legado, no gerenciamento de clientes da empresa ECOS Sistemas. Este documento trata-se de um relatório final de estágio supervisionado realizado na empresa ECOS Sistemas, com duração de 360 horas sob a supervisão do Alisson Marcel, diretor de desenvolvimentos da empresa.

O capítulo 1 consiste na introdução do documento e sua contextualização histórica. O problema e os impactos causados à empresa onde o estágio foi realizado são expostos no capítulo 2. Os objetivos que se desejam atingir com a realização do trabalho são enunciados no capítulo 3. O capítulo 4 trata-se de justificar a importância da solução encontrada. Na Fundamentação Teórica, são listadas e descritas as tecnologias usadas para implementar a solução. O método usado para o desenvolvimento da solução é descrito no capítulo 6. Os diagramas e as funcionalidades do sistema desenvolvido são mostrados no capítulo 7. A análise do sistema e as dificuldades encontradas durante o desenvolvimento do mesmo são

informadas nas considerações finais. Sugestões de trabalhos futuros são encontradas no capítulo 9.

2. PROBLEMA

A gestão de clientes é uma das atividades mais corriqueiras nas empresas que desenvolvem *softwares*. Dependendo da quantidade de clientes e do sistema utilizado para geri-los, essa atividade pode ser algo simples, ou complexo, de ser realizado.

Ao gerenciar os seus clientes, a empresa tem como objetivo manter informações atualizadas, a fim de controlar a prestação de serviços fornecida a cada um deles à luz do que foi previamente estabelecido em contrato.

A ECOS Sistemas, uma empresa de pequeno porte com matriz localizada em Mossoró-RN e uma filial localizada em Natal-RN, conta com 550 clientes, aproximadamente, distribuídos em alguns estados da região Nordeste do Brasil.

A empresa em si já conta com um sistema interno de gestão de clientes, sob a forma de um *software Desktop*. Esse sistema está escrito na versão 3 da linguagem *Delphi*, 20 (vinte) versões atrás da versão atual da linguagem (*Embarcadero Delphi XE6*[1]). O sistema de gerenciamento utilizado pela empresa é considerado um sistema legado, ou seja, é um sistema antigo, que utiliza um banco de dados obsoleto e de difícil manutenção. Essa dificuldade na manutenção impede que o sistema seja migrado para uma versão mais atual da linguagem de programação *Delphi*.

O sistema funciona com duas bases de dados: uma localizada em Mossoró e outra localizada em Natal. O uso desse sistema de gestão, junto com o fato de haver essas duas bases de dados, resulta na seguinte problemática: é necessário ficar atualizando diariamente as informações contidas nos dois bancos de dados, ou seja, é feito um *backup* das informações contidas no banco de dados de Natal e depois é efetuada a transferência das mesmas para o banco de dados de Mossoró, e vice-versa. A forma como é feita a atualização das informações, hoje, consome muito tempo. Ultimamente, esses *backups* são feitos com a intenção de não deixar nenhum cliente sem manutenção por parte da empresa, independente do cliente entrar em contato com a matriz ou com a filial.

[1] https://www.embarcadero.com/br/products/delphi

3. OBJETIVOS

Este trabalho descreve a solução desenvolvida para o problema das bases de dados duplas com as quais a ECOS Sistemas opera.

3.1. OBJETIVO GERAL

Desenvolver uma solução em forma de um sistema WEB, que trabalhe com um banco de dados único e que gerencie os clientes da empresa, e implantar esse sistema na matriz e filial, após a realização de testes funcionais e estruturais.

3.2. OBJETIVOS ESPECÍFICOS

- Criar uma interface gráfica para o sistema, utilizando tecnologias atuais em *design* gráfico;
- Iniciar a migração das funções básicas encontradas no *software desktop* da empresa para o sistema WEB a ser desenvolvido. Tais funções são:
  - Cadastrar Cliente, Sistema e Empresa;
  - Editar Cliente, Sistema e Empresa;
  - Listar Cliente, Sistema e Empresa;
  - Remover Cliente, Sistema e Empresa;
  - Buscar Cliente, Sistema e Empresa;
  - Gerar relatório contendo informações acerca do suporte fornecido para a empresa-cliente.
- Transferir as informações contidas nos dois bancos de dados da empresa para um único banco de dados;
- Identificar os testes funcionais e estruturais que deverão ser realizados para efetivar a implantação do sistema na empresa.

4. JUSTIFICATIVA

Ultimamente, a empresa ECOS Sistemas lida diariamente com *backups* das informações contidas na matriz e filial, e isso despende um tempo substancial do programador da empresa. O sistema também está escrito em uma linguagem de programação obsoleta (*Delphi* 3), tornando-o, assim, um sistema legado, de difícil manutenção.

A migração, ou conversão, do *software desktop* para um sistema WEB fará com que haja uma melhoria na atividade de gestão de clientes da empresa.

Primeiramente, o banco de dados será unificado, ou seja, as informações se localizarão em um só banco de dados e serão atualizadas de forma automática, via Internet. Com isso, a matriz e a filial irão dispor das mesmas informações em tempo real e deixarão de fazer os *backups* diários. Assim sendo, não terão mais nenhum desperdício de tempo.

Segundo, o sistema deixará de ser um sistema legado, pois será migrado para uma linguagem de programação (*Groovy*) atual e usará um *framework* (*Grails*) também considerado atual. Também disporá de uma interface gráfica mais amigável e intuitiva.

E, por fim, os clientes poderão ser atendidos em qualquer uma das empresas ECOS (matriz e filial) sem se preocupar com o fato de não terem o suporte fornecido, haja vista que as informações sobre eles não estariam contidas, ou atualizadas, em algum dos dois bancos de dados.

5. FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa descrever as tecnologias que foram utilizadas para o desenvolvimento do sistema WEB proposto. As imagens e os trechos de código fornecidos neste capítulo são de autoria própria e foram extraídos a partir do código-fonte do sistema desenvolvido.

Os *frameworks Grails* e *Bootstrap*, atuando em conjunto, garantem a alta-produtividade no desenvolvimento do sistema. O primeiro assegura que as funcionalidades básicas (cadastrar, listar, editar, remover, etc.) sejam implementadas de forma automática e o segundo propicia que as interfaces gráficas referentes a cada uma dessas funcionalidades sejam desenvolvidas utilizando o reúso de código.

A linguagem de programação *Groovy*, ao conter as características básicas da linguagem de programação *Java*, tornará a programação do sistema mais simples e intuitiva. O ambiente de desenvolvimento *Groovy/Grails Tool Suite* (GGTS) que é propriamente voltado para a utilização do *framework Grails* em conjunto com a linguagem de programação *Groovy*, será o elo que tornará possível a comunicação entre os dois. A popularidade e a facilidade de manutenção fez com que o *MySQL* fosse o banco de dados escolhido para a aplicação.

A seção sobre o *framework Grails* descreve a utilidade, o modelo de desenvolvimento com o qual este *framework* trabalha e como ele garante uma parte da alta-produtividade. A seção da linguagem de programação *Groovy* contextualiza-a historicamente, evidenciando as suas características próprias, explanando as comparações com *Java* e mostrando exemplos de sua utilização. A seção 5.3 informa como o ambiente GGTS está organizado, quais as funções das principais seções que o compõem e como ele faz para que a comunicação entre a linguagem de programação *Groovy* e o *framework Grails* seja possível, exibindo exemplos de códigos. A seção 5.4 demonstra como o *Bootstrap* garante outra parte da alta-produtividade, salientado quem são os seus criadores/desenvolvedores e como é efetuada a sua configuração no ambiente GGTS. Sobre a seção do banco de dados *MySQL*, denotam-se os motivos da escolha e suas características particulares.

5.1. FRAMEWORK GRAILS

Grails é um *framework* de alta produtividade, desenvolvido em código aberto, graças à utilização do paradigma da programação por convenção que preserva o desenvolvedor dos

detalhes de configuração. Ele é comumente utilizado na construção de aplicações para a WEB fazendo uso da linguagem de programação *Groovy*, uma linguagem baseada no *Java*.

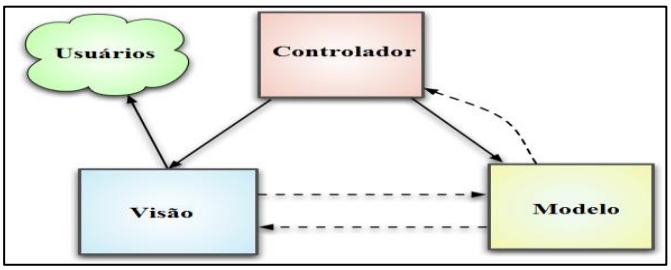
Entende-se como programação por convenção, ou convenção sobre configuração, a ideia do programador precisar definir apenas os aspectos não convencionais da aplicação. Segundo Fernandes (2010), "tal conceito prega que o desenvolvedor não precisa dizer, ou melhor, configurar, como o *framework* irá se comportar toda vez que for construir uma aplicação". Um exemplo disso é que não é necessário criar arquivos de mapeamento Objeto-Relacional, pois eles já estão embutidos na configuração básica do *framework*. Outro exemplo é que se houver uma classe chamada *Cliente*, a tabela correspondente no banco de dados é chamada de *clientes* por padrão.

O *framework Grails* utiliza o padrão de desenvolvimento *Model-View-Controller* (MVC), que é um modelo de arquitetura de *software* que separa a representação da informação da interação do usuário com ele.

As ideias centrais do MVC compreendem a reutilização de código e a separação de conceitos. Seguindo a linha de pensamento do MVC, o modelo (*model*) é a parte na qual os dados da aplicação, regras de negócios, lógica e funções são definidos. Uma visão (*view*) pode ser qualquer saída de representação dos dados, como uma tabela, um diagrama ou até mesmo uma interface gráfica. É possível ter várias visões do mesmo dado. O controlador (*controller*) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão.

Na Figura 1, é mostrado um diagrama exemplificando a relação e os comportamentos observados entre as representações do Modelo, Visão e Controle. O usuário se comunica com a aplicação através da interface gráfica (visão). O controlador, ao receber a requisição do usuário, informa ao modelo quais são os dados necessários para a finalização da requisição. O modelo retorna ao controlador os dados requisitados. O controlador exibe as informações dos dados para o usuário através da visão.

Figura 1 – Diagrama de representação das relações entre *Model*, *View* e *Controller*



Fonte: www.mrcandrefarias.blogspot.com.br/2013/09/porque-mvc-nao-e-tres-camadas.html

Além do citado anteriormente, o *Grails* também fornece *templates Web* para fácil implementação da interface com o usuário. Esses *templates Web* são interfaces gráficas customizáveis de acordo com a proposta do desenvolvedor. Por exemplo, usando o *prompt* de comando nativo do ambiente GGTS, podem-se fazer *downloads* de várias interfaces de telas de *login*. Essas telas se adequam de acordo com o projeto da aplicação que está sendo desenvolvida.

O *framework Grails* é utilizado durante todo o desenvolvimento da aplicação, no que compreende a definição das interfaces gráficas, a utilização do modelo MVC e as configurações necessárias ao funcionamento do sistema.

5.2. LINGUAGEM DE PROGRAMAÇÃO GROOVY

Groovy é uma linguagem de programação orientada a objetos desenvolvida para a plataforma *Java* como alternativa à linguagem de programação *Java*. Foi criada em 2003 pelo programador James Strachan.

A descrição presente no *site* oficial[2] da linguagem diz:

> "*Groovy* é uma linguagem ágil e dinâmica para a plataforma *Java* com muitas características que são inspiradas por linguagens como *Python*, *Ruby* e *Smalltalk*, trazendo uma sintaxe *Java-like* para os desenvolvedores *Java*. Segundo o *site* oficial da linguagem *Groovy*." (LAFORGE, 2011).

Aliam-se à linguagem *Groovy* características encontradas na linguagem *Java*, sendo algumas delas: é compilada dinamicamente para a Máquina Virtual *Java* e interopera com outros códigos e bibliotecas *Java*.

Uma característica encontrada na linguagem *Groovy* que não se encontra em *Java* padrão é a possibilidade de fazer a tipagem dos dados tanto de modo estático quanto de modo dinâmico. Na tipagem estática, o programador define o tipo daquela variável que está sendo declarada. Por outro lado, na tipagem dinâmica não é exigida nenhuma declaração do tipo de dado; a própria linguagem de programação é capaz de escolher que tipo utilizar dinamicamente para cada variável.

[2] http://groovy.codehaus.org

A Listagem 1 mostra um exemplo de tipagem estática e tipagem dinâmica na linguagem de programação *Groovy*. Observe que não são utilizados os métodos *get* e *set*. O uso do ponto-e-vírgula, ao fim da declaração de cada variável, é opcional.

Listagem 1 – Exemplo do uso das tipagens estática e dinâmica na linguagem *Groovy*

```
package gestaoclientes

class Cliente {

    def nomeCompleto //tipagem dinâmica sem atribuição de valor à variavel
    int idade //tipagem estática sem atribuição de valor à variável

    //o uso de ponto-e-vírgula é opcional
    String outroNome = "Alisson"; //tipagem estática com atribuição de valor à variavel
    def outraIdade = 23; //tipagem dinâmica com atribuição de valor à variável

}
```

Segundo ALBUQUERQUE (2011), "o *Groovy* possibilita ao desenvolvedor não precisar declarar classes e métodos, permitindo executar o código em um escopo global". A Listagem 2 ilustra um exemplo de código, em *Java*, que imprime palavras com 4 ou menos letras a partir de um vetor de *strings* pré-definido e a Listagem 3 exibe o mesmo exemplo na linguagem *Groovy*. Como dá pra perceber ao comparar as duas listagens, o código escrito em *Groovy* contém bem menos linhas e é bem mais simples que o código escrito em *Java*.

Listagem 2 – Trecho de código escrito em *Java*

```
class Filtro {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Alfinete", "Sabrina", "Zina");
        List<String> shorts = new ArrayList<String>();
        for(String item : list) if(item.length() <= 4) shorts.add(item);
        for(String item : shorts) System.out.println(item);
    }
}
```

Fonte: autoria própria

Listagem 3 – Trecho de código escrito em *Groovy*

```
class Filtro {

    list = ["Alfinete", "Sabrina", "Zina"]
    shorts = list.findAll { it.size() <= 4 }
    shorts.each { println it }

}
```

Fonte: autoria própria

JUDD (2008) afirma que, a sintaxe do *Groovy* é muito mais flexível se comparada ao *Java*. Dezenas de linhas de código na linguagem *Java* podem ser reduzidas para algumas linhas de código em *Groovy*.

Atualmente, a linguagem *Groovy* é uma especificação do *Java Community Process* (JCP) (*Java Specification Requests* (JSR) 241), sendo considerada a segunda linguagem oficial da plataforma Java (LAFORGE, 2004).

5.3.AMBIENTE DE DESENVOLVIMENTO GROOVY/GRAILS TOOL SUITE

A descrição presente no site oficial[3] do *Groovy/Grails Tool Suite* (GGTS) diz:

> "O *Groovy/Grails Tool Suite*™ (GGTS) fornece o melhor ambiente de desenvolvimento baseado em *Eclipse* para construir aplicativos usando *Groovy* e *Grails*. O GGTS fornece suporte para as ultimas versões do *Groovy* e *Grails*, e vem no topo dos mais recentes lançamentos do *Eclipse*."

O ambiente GGTS utiliza o padrão de desenvolvimento MVC, conforme exibido na Figura 2. Vale salientar que, no GGTS, o modelo, a visão e o controlador são chamados de *Domain*, *Views* e *Controllers*, respectivamente.

Figura 2 – Disposição do padrão de desenvolvimento MVC no projeto *GestaoClientesNew*

Fonte: autoria própria

Na seção denominada *Domain*, são criadas as classes de domínio. Nas classes de domínio estarão os dados (variáveis) que serão utilizados pelas classes de controle (*Controllers*). Em outras palavras, na classe de domínio são declaradas as variáveis do sistema e as suas propriedades de armazenamento no banco de dados.

A Listagem 4 ilustra um exemplo do uso de uma classe de domínio chamada Cliente. Na primeira parte são declaradas as variáveis utilizadas no sistema. Na segunda parte, identificada como *Constraints*, são definidas as propriedades de armazenamento das variáveis

[3] https://grails.org/products/ggts

no banco de dados. Como podemos ver na Listagem 4, a variável *codigo*, do tipo inteiro, terá valor único e não poderá ter o valor do seu campo em branco ou nulo, ou seja, é obrigatória a inserção de qualquer valor diferente de zero.

Listagem 4 – Exemplo de uma classe de domínio

```
package gestaoclientes

import java.util.Date;

class Cliente {

    int codigo
    String nome
    String cpfCnpj
    String nomeFantasia
    String endereco
    Date dataDeNascimento

    static constraints = {
        codigo(nullable:false, blank:false, unique:true)
        nome(nullable:false, blank:false)
        cpfCnpj(nullable:false, blank:false, cpfCnpj:true, unique:true)
        nomeFantasia(nullable:true)
        endereco(nullable:false, blank:false)
        dataDeNascimento(nullable:true)
    }
}
```

O GGTS permite a instalação de *plugins*, criados por terceiros, que possam validar CPFs, CNPJs, e-mails, etc. Para usá-los, basta instalá-los e habilitar o uso nas *constraints*. Na Listagem 4, na variável *cpfCnpj*, é visto um exemplo de utilização de um *plugin* que valida tanto CPF quanto CNPJ. O valor do campo CPF/CNPJ só é armazenado no banco de dados se for validado pelo *plugin*.

A classe de domínio Cliente, depois de criada, é nomeada como *Cliente.groovy*. Esse padrão de nomeação serve para qualquer classe criada na seção *domain*.

Na seção *controllers*, são criados e definidos os controles de cada classe de domínio. Por padrão, cada classe de controle será denominada por meio do nome da classe de domínio acrescida da palavra *Controller*, para especificar que é uma classe de controle referente àquela classe de domínio.

É mostrada, na Listagem 5, a utilização da classe de controle chamada ClienteController. Nessa é habilitado o *scaffold*, que permite ao programador especificar como o banco de dados da aplicação poderá ser usado. O *scaffold*, quando habilitado, implementa de forma automática os métodos *create*, *read*, *update* e *delete* para as entradas do

banco de dados. Junto a essa implementação, também são gerados os *Web Templates* para cada um desses métodos.

Listagem 5 – Uma classe de controle utilizando o *scaffold*

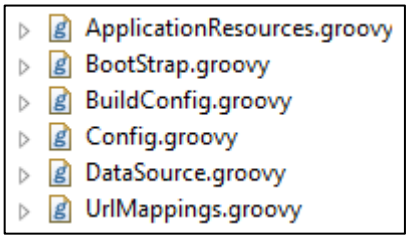
```
package gestaoclientes



import static org.springframework.http.HttpStatus.*
import grails.transaction.Transactional
import grails.plugin.springsecurity.annotation.*
import grails.plugin.springsecurity.Secured.*


class ClienteController {

    static scaffold = true
}
```

Grande parte da alta produtividade garantida pelo uso do *framework Grails* se encontra na habilitação e utilização do *scaffold*, pois, ao habilitá-lo, ele gera as funções cadastrar, editar, remover e listar para cada classe de controle.

Na seção *Views*, são armazenados os *Web Templates* gerados pela utilização do *scaffold*. Quando são geradas as funções cadastrar, editar, remover e listar, são criadas, junto a elas, interfaces gráficas, que também são chamadas de *Web Templates*, para cada uma dessas funções. Essas interfaces gráficas são armazenadas em diretórios contidos na seção *Views.*

No ambiente GGTS, há uma seção de configuração denominada *conf*. Nela, há os arquivos necessários para a configuração da conexão com o banco de dados, o controle de acesso às páginas do sistema desenvolvido, os papéis e os usuários criados pelo uso do *BootStrap* e as instalações dos *plugins*. A Figura 3 mostra os arquivos que estão presentes na seção *conf*.

Figura 3 – Arquivos de configuração encontrados na seção *conf*



Fonte: autoria própria

No arquivo de configuração *BootStrap.groovy*, cria-se o serviço de segurança do sistema. Este serviço consiste no seguinte: ao ser definido, é criado dois papéis de usuário: o papel usuário e o papel administrador. Esses papéis consistem em controlar o acesso que cada usuário terá durante o uso do sistema. O papel administrador garante acesso irrestrito ao usuário. Entretanto, o papel usuário limitará o acesso do usuário a apenas algumas funções. O serviço de segurança também define uma tela de *login* e uma conta de administrador padrão.

O arquivo *BuildConfig.groovy* diz respeito às configurações de repositórios, dependências e *plugins* utilizados no desenvolvimento das aplicações. Um exemplo de utilização desse arquivo de configuração é a definição do uso do *plugin* de validação de CPFs e CNPJs. O *plugin*, que faz essa validação, está identificado como *":br-validation:0.3"*. Na Listagem 6, é apresentado o trecho de código no qual são definidos os *plugins* utilizados na aplicação. É importante observar que, toda vez que um novo *plugin* é inserido, é necessário recompilar o projeto.

Listagem 6 – Trecho de código contendo os *plugins* utilizados no sistema

```
// plugins for the compile step
compile ":scaffolding:2.0.2"
compile ':cache:1.1.1'
compile ':spring-security-core:2.0-RC2'
compile ":spring-security-ui:1.0-RC1"
compile ":mail:1.0.4"
compile ":br-validation:0.3"
```

No arquivo de configuração *Config.groovy*, é utilizado o controle de acesso para definir qual dos usuários terá acesso a quais páginas do sistema. Esse controle é definido por regras (LEDBROOK, 2010). Segue o nome das regras e o que cada uma delas significa:

- IS_AUTHENTICATED_ANONYMOUSLY: qualquer um tem acesso; não é necessário o usuário se registrar no sistema;
- IS_AUTHENTICATED_REMEMBERED: apenas usuários conhecidos que tenham acessado ou que são lembrados a partir de uma sessão anterior tem acesso permitido;
- IS_AUTHENTICATED_FULLY: os usuários têm que se registrar no sistema para ganhar acesso, mesmo que eles tenham ativado a caixa de checagem “lembre-me” na última vez em que acessaram o sistema.

As regras supracitadas podem ser utilizadas em conjunto com outras regras definidas pelo programador, a saber:

- ROLE_USER: os usuários que tem o papel usuário associado ao seu nome de usuário tem acesso permitido;
- ROLE_ADMIN: os usuários que tem o papel administrador associado ao seu nome de usuário tem acesso permitido.

O código exibido na Listagem 7 expressa o controle de acesso nas páginas do sistema desenvolvido. Percebe-se que a mesma regra se aplica a todas as páginas. Tais regras definem que qualquer usuário pode ter acesso às páginas e funções implementadas no sistema, desde que o mesmo tenha se autenticado previamente.

Listagem 7 – Controle de acesso definido para as páginas do sistema

```
// Added by the Spring Security Core plugin:
grails.plugin.springsecurity.userLookup.userDomainClassName = 'gestaoclientes.User'
grails.plugin.springsecurity.userLookup.authorityJoinClassName = 'gestaoclientes.UserRole'
grails.plugin.springsecurity.authority.className = 'gestaoclientes.Role'
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
    '/':                              ['IS_AUTHENTICATED_REMEMBERED'],
    '/cliente/**':                    ['IS_AUTHENTICATED_REMEMBERED'],
    '/empresa/**':                    ['IS_AUTHENTICATED_REMEMBERED'],
    '/sistema/**':                    ['IS_AUTHENTICATED_REMEMBERED'],
    '/sistemasecos/**':               ['IS_AUTHENTICATED_REMEMBERED'],
    '/index':                         ['IS_AUTHENTICATED_REMEMBERED'],
    '/index.gsp':                     ['IS_AUTHENTICATED_REMEMBERED'],
    '/**/js/**':                      ['IS_AUTHENTICATED_REMEMBERED'],
    '/**/css/**':                     ['IS_AUTHENTICATED_REMEMBERED'],
    '/**/images/**':                  ['IS_AUTHENTICATED_REMEMBERED'],
    '/**/favicon.ico':                ['IS_AUTHENTICATED_REMEMBERED']
]
```

O *DataSource.groovy* define as configurações gerais acerca do banco de dados. Nela, é definido o *drive* do banco de dados, a *Uniform Resource Locator* (URL) onde se encontra o banco de dados daquela aplicação e o comportamento do banco de dados que é definido pelas seguintes regras:

- *create:* cria o banco de dados se ele não existir, mas não o modifica caso ele exista. Exclui os dados existentes;
- *create-drop:* destrói e recria o banco de dados quando o *Grails* é executado;
- *update:* cria o banco de dados, caso não exista; modifica-o, caso ele exista.

O GGTS é considerado um ambiente de desenvolvimento completo, dada à facilidade de integração com *plugins* de terceiros, ao seu uso no desenvolvimento de aplicações que trabalham com a linguagem *Groovy* e o *framework Grails* e à praticidade de se lidar com o controle de acesso às páginas. O uso desse ambiente é justificado por ser a única *Integrated Development Environment* (IDE) gratuita, de código aberto e totalmente voltada para o desenvolvimento de aplicações que utilizem *Groovy* e *Grails*.

5.4. FRAMEWORK BOOTSTRAP

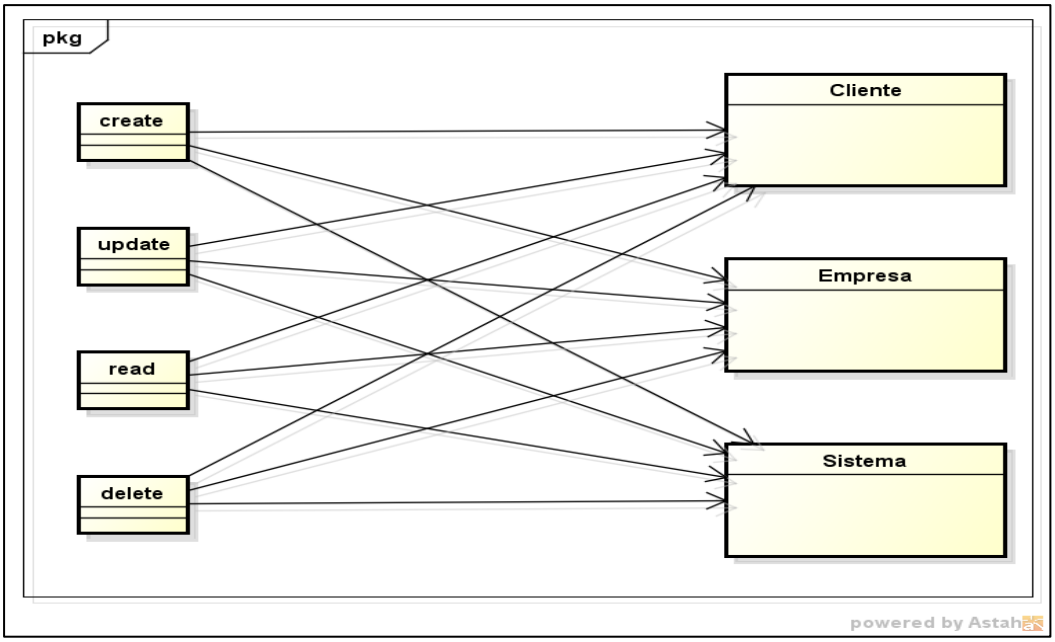
Conhecido originalmente como *Framework Twitter Bootstrap*, seu objetivo é facilitar o desenvolvimento de interface (*front-end*) para páginas WEB, usando da disponibilização dos elementos *HyperText Markup Language* (HTML) mais utilizados, além de elementos personalizados com o uso de classes *Cascading Style Sheets* (CSS) padrões (HTML5DEV, 2013). Entende-se como *front-end*, a interface gráfica voltada para a comunicação com o usuário.

O *Bootstrap* foi criado por Mark Otto e Jacob Thornton, ambos engenheiros do *Twitter*.

Como visto anteriormente, quando habilitamos o uso do *scaffold* nas classes de controle, definimos interfaces gráficas padrões para os métodos *create, read, update* e *delete.* A importação dos arquivos do *Bootstrap* para o desenvolvimento da aplicação faz com que as interfaces padrões a serem usadas para esses métodos passem a ser as interfaces padrões definidas pelo *Framework Bootstrap.*

O *Bootstrap* também garante uma parte da alta produtividade. É mostrado na Figura 4, um diagrama que representa as associações entre os métodos gerados pelo *scaffold* e as classes de domínio. O objetivo da figura é evidenciar que, por exemplo, se alterarmos a interface gráfica do método *create*, então a mesma alteração será aplicada em todos os métodos *create* associados a cada classe domínio. O mesmo acontece se as interfaces dos outros métodos forem alteradas.

Figura 4 – Diagrama mostrando as associações entre os métodos e as classes de domínio



Fonte: autoria própria

Como visto nas seções anteriores, o *framework Grails* garante uma parte da alta-produtividade ao implementar, de forma automática, as funções básicas do sistema: cadastrar, listar, editar e remover. O *Grails* trata da parte funcional do sistema. O *framework Bootstrap*, por sua vez, garante outra parte da produtividade ao criar, também de forma automática, as interfaces gráficas referentes a cada uma das funções básicas geradas pelo *Grails*. O *Bootstrap* trata da parte da comunicação com o usuário através de interfaces gráficas (visão).

Ao aliar o uso desses dois *frameworks*, em torno de 1 (uma) semana, no máximo, a equipe de desenvolvimento já está apta para mostrar ao cliente uma versão do sistema com as funcionalidades básicas implementadas e com uma interface gráfica amigável. Isso explica o porquê foi escolhido utilizar os dois durante o desenvolvimento do sistema: a velocidade na entrega de uma versão mais básica, porém funcional, do sistema.

5.5. BANCO DE DADOS MYSQL

O *MySQL*[4] foi criado na Suécia por David Axmark, Allan Larsson e Michael Widenius. Este banco de dados utiliza a linguagem SQL (Linguagem de Consulta Estruturada,

[4] http://www.mysql.com

do inglês *Structured Query Language*) como interface. É, atualmente, um dos bancos de dados mais populares, com mais de 10 (dez) milhões de instalações pelo mundo.

Destacam-se as seguintes características do *MySQL*:

- É um banco de dados multiplataforma;
- Tem um bom desempenho e estabilidade;
- Pouco exigente quanto a recursos de novos *hardwares*;
- Replicação facilmente configurável.

No GGTS, a conexão com o banco de dados se dá pelo arquivo de configuração *DataSource.groovy*. Na Listagem 8, é visualizado como se encontra a configuração do banco de dados utilizado no sistema. Nota-se que a listagem enfatiza a regra que define o tipo de banco de dados e o diretório de sua localização.

Listagem 8 – Trecho de código extraído do arquivo de configuração *DataSource.groovy*

```
development {
    dataSource {
        dbCreate = "update" // one of 'create', 'create-drop', 'update', 'validate', '
        url = "jdbc:mysql://localhost:3306/gestaoclientesnew"
    }
}
```

O *MySQL* foi escolhido pois é gratuito, de fácil instalação, não é complicado fazer a integração com o ambiente GGTS e o desenvolvedor (no caso, o autor) já sabia trabalhar com o mesmo de forma mais eficiente que com outros Sistemas de Gerenciamento de Banco de Dados (SGBDs).

6. METODOLOGIA DE DESENVOLVIMENTO DO SISTEMA

O *Scrum* é um processo de desenvolvimento iterativo e incremental para o gerenciamento de projetos e desenvolvimento ágil de software. Este método foi o escolhido para gerenciar o desenvolvimento do sistema proposto.

No *Scrum*, os projetos são divididos em ciclos (tipicamente mensais) chamados de *Sprints*. O *Sprint* representa um conjunto de atividades que devem ser executadas em um dado período de tempo. As metodologias ágeis de desenvolvimento de *software* são divididas em iterações, que são chamadas de *Sprints*, no caso do *Scrum* (SCRUM, 2014).

As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*. O *Product Backlog* é análogo ao processo de Levantamento de Requisitos.

A Tabela 1 mostra um exemplo salientando os dados extraídos do *Product Backlog* referente ao sistema desenvolvido. Ao definirmos uma prioridade para cada funcionalidade, conseguimos definir quais funcionalidades serão implementadas primeiro. A estimativa nos dá uma noção de quanto tempo será demandado dos desenvolvedores para implementar aquela função.

Tabela 1 - Trecho do *Product Backlog* para o desenvolvimento do sistema proposto

PRODUCT BACKLOG					
ID	NOME	PRIORIDADE	ESTIMATIVA	DEMONSTRAÇÃO	NOTAS
1	Cadastro de Empresas	Normal	2 dias	Deve se logar, abrir a aba de Cadastros, clicar em Empresas, informar os dados da Empresa (Código, Nome de Acesso, Razão Social, Endereço, Telefone, etc.) e depois confirmar os dados.	
2	Cadastro de Clientes	Normal	2 dias	Deve se logar, abrir a aba de Cadastros, clicar em Clientes, informar os dados do Cliente (Código, Nome, Nome Fantasia, Endereço, Telefone, etc.) e depois confirmar os dados.	
3	Cadastro de Sistemas	Normal	2 dias	Deve se logar, abrir a aba de Cadastros, clicar em Sistemas, informar os dados do Sistema (Sistema, Número de Série, Versão, Tipo Versão, Usuário, etc.) e depois confirmar os dados.	
4	Contato com o Cliente	Alta	3 dias	Deve-se logar, realizar a busca pelo nome do Cliente e exibir os dados do Cliente na tela (quais os softwares adquiridos pelo cliente, se há ou não manutenção para o mesmo, verificar alguma pendência de pagamento, etc.).	

Fonte: autoria própria

As funções encontradas no *Scrum* são as seguintes:

- *Scrum Master* (mestre *Scrum*)*:* é o facilitador, ou seja, é o que remove os impedimentos à capacidade da equipe para entregar o objeto gerado pelo *Sprint.* Ele não é o líder da equipe, porém, é normalmente comparado a um Gerente de Projeto;
- *Product Owner* (dono do produto): representa a voz do cliente e é responsável por garantir que a equipe agregue valor ao negócio. Em suma, é o responsável por transmitir aos desenvolvedores os requisitos exigidos pelo cliente;
- *Development Team* (equipe de desenvolvimento): são os responsáveis pela entrega do produto. Normalmente é composta por 3 a 7 pessoas com habilidades multifuncionais (analisar, projetar, desenvolver, documentar, etc.). É recomendada que ela seja auto-organizada e auto-induzida.

O dono do produto é responsável por transmitir os requisitos para a equipe de desenvolvimento. Se algum requisito for modificado ou inserido por parte do cliente, o dono do produto é, também, o responsável por informar à equipe de desenvolvimento. A equipe de desenvolvimento é responsável por implementar as funcionalidades/requisitos em cada *Sprint.* A equipe de desenvolvimento também é responsável pela entrega do produto final. Durante a realização de cada *Sprint*, o mestre *Scrum* fica incentivando a equipe de desenvolvimento e removendo qualquer empecilho (falta de equipamentos, ausência de pessoas na equipe por motivos pessoais, desmotivação da equipe, etc.) que possa causar algum atraso na entrega do produto.

Os *Sprints* são definidos através de uma reunião de planejamento, conhecida como *Spring Planning Meeting*, na qual o dono do produto (*Product Owner*) prioriza os itens do *Product Backlog* e a equipe de desenvolvimento seleciona as atividades que será capaz de implementar durante o *Sprint* que se inicia. As tarefas alocadas em um *Sprint* são transferidas do *Product Backlog* para o *Sprint Backlog*. Este último consiste em uma planilha contendo apenas as funcionalidades que serão desenvolvidas durante aquele *Sprint.*

A cada dia de uma *Sprint*, a equipe faz uma breve reunião (normalmente de manhã), chamada *Daily Scrum Meeting*. O objetivo é disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.

Ao final de um *Sprint*, a equipe de desenvolvimento apresenta as funcionalidades implementadas. Depois, é feita uma retrospectiva do que já foi implementado até agora e a equipe de desenvolvimento parte para o planejamento do próximo *Sprint*.

A Figura 5 exibe todos os processos envolvidos durante o desenvolvimento do método *Scrum.* Primeiro, cria-se o *Product Backlog*. Depois, é definido o *Sprint* com as funcionalidades a serem desenvolvidas. Cada *Sprint* dura, em média, de 2 a 4 semanas. Durante o período de desenvolvimento do *Sprint*, é feita uma reunião diária para saber o que foi feito até agora pela equipe de desenvolvimento. Após o fim de cada *Sprint*, é feita outra reunião para definir o próximo *Sprint*. O produto final é entregue depois de feitas as implementações de todas as funcionalidades contidas no *Product Backlog*.

Figura 5 – Processos envolvidos durante a realização do método *Scrum*

Fonte: http://blog.opus-software.com.br/scrum-ciclos-de-valor-3/

Para auxiliar na utilização do *Scrum*, usou-se uma ferramenta WEB desenvolvida por Luiz Fernando Virgínio da Silva[5] e totalmente gratuita. Tal ferramenta se chama *ScrumIt*![6].

A escolha pela utilização do método *Scrum* se deu pelo fato de ser necessário o envolvimento de apenas duas pessoas durante o desenvolvimento do sistema: eu como a equipe de desenvolvimento e o Alisson Marcel, supervisor do estágio e diretor de desenvolvimento da empresa ECOS Sistemas, como o *Product Owner* e o *Scrum Master*.

A opção pelo *Scrum* se deu também por se tratar de um método ágil no desenvolvimento de *software*, por conter pouca documentação para preencher, por ser possível modificar as prioridades dos *Sprints*, garantindo que os que não foram finalizados possam ser alterados sem dificuldades e por reduzir problemas como falta de planejamento, mudança constante de requisitos e a falta de participação do cliente.

[5] Graduado em Sistemas de Informação e especialista em Desenvolvimento de Sistemas para a WEB, ambos pela UnP, segundo consta na descrição do perfil de usuário na comunidade do ScrumIt!
[6] www.scrumit.com.br

7. ECOS MANAGER INTERNAL (EMI)

Neste capítulo será descrito o sistema que foi proposto como solução para o problema encontrado na empresa ECOS Sistemas. Serão mostrados, além das funcionalidades do sistema, alguns diagramas em *Unified Modeling Language* (UML) para proporcionar um entendimento melhor ao leitor**.**

A seção 7.1 tratará da descrição dos diagramas UML e a seção 7.2 explanará o sistema em si: as funcionalidades, as telas (interfaces gráficas) que o compõe e a forma como foi implementada cada uma delas.

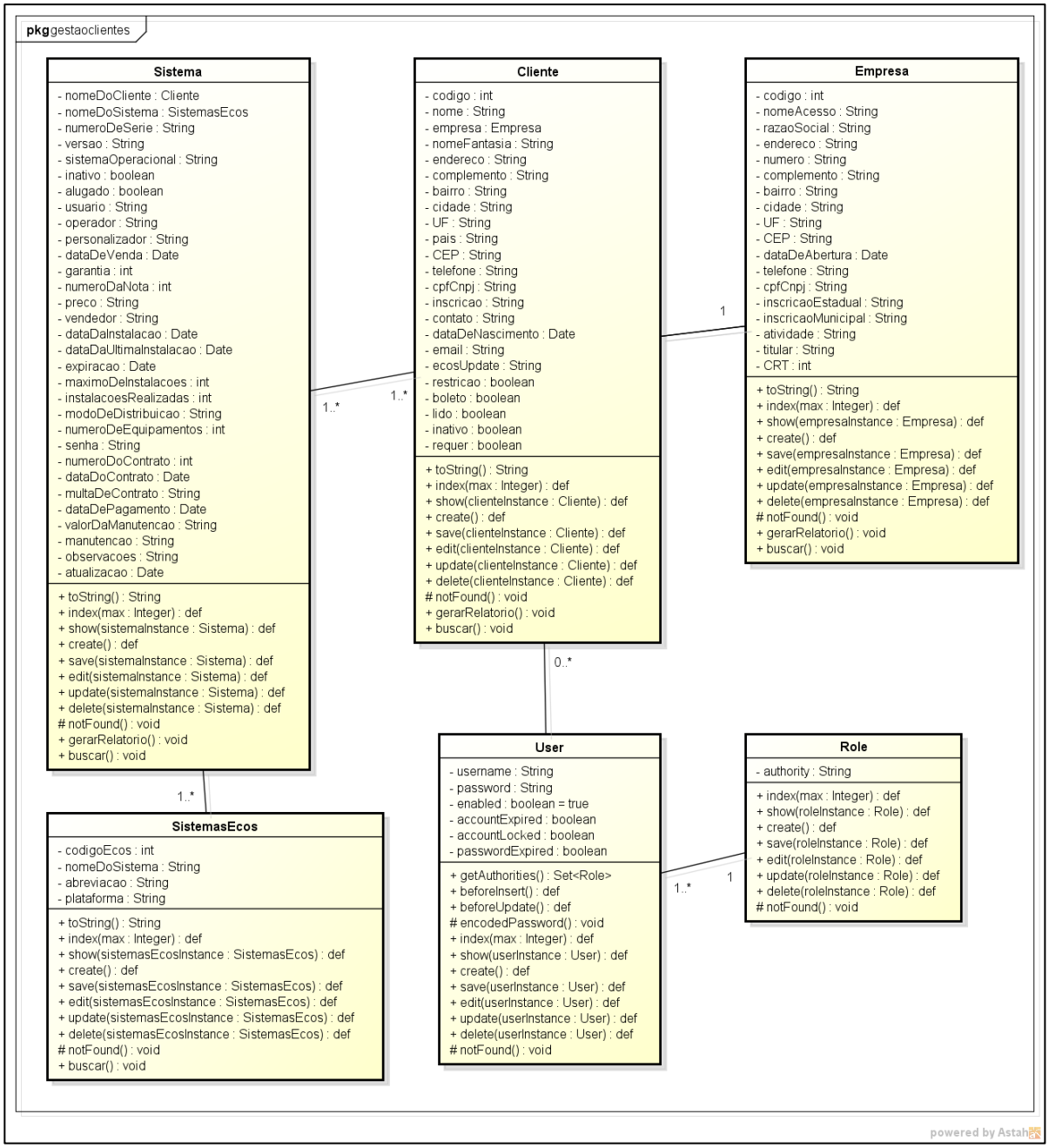
7.1. DIAGRAMAS UML

Os diagramas UML permitem que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados. Junto com a notação gráfica, a UML também permite especificar significados. Os objetivos da UML são: especificação, documentação e estruturação para uma visualização lógica do desenvolvimento completo de um sistema de informação.

O sistema, antes de ser implantado efetivamente, irá funcionar apenas com usuários administradores, ou seja, os usuários terão acesso às funcionalidades encontradas no sistema. O EMI contará com as seguintes classes: Cliente, Empresa, Sistema, SistemasEcos, *Role* e *User*.

A Figura 6 mostra o diagrama de classes do sistema EMI. Este diagrama tem como objetivo mostrar as classes que o compõe e os relacionamentos entre elas. Para um usuário (classe *User*) existe um papel (classe *Role*), e para um papel existem um ou mais usuários. Um cliente pertence a uma empresa e esse cliente pode ter um ou mais diferentes sistemas. Um sistema poderá pertencer a um ou mais clientes. Um usuário poderá ter zero ou mais clientes cadastrados. Na Figura 6, percebe-se que existe a classe Sistema que faz o intermédio entre as classes Cliente e SistemasEcos. Essa classe é o elo que faz com que um sistema (classe SistemaEcos) pertença a vários clientes (classe Cliente) e um cliente tenha vários sistemas diferentes (classe SistemasEcos). A classe SistemaEcos é a responsável por armazenar as informações de cada um dos sistemas desenvolvidos pela ECOS Sistemas.

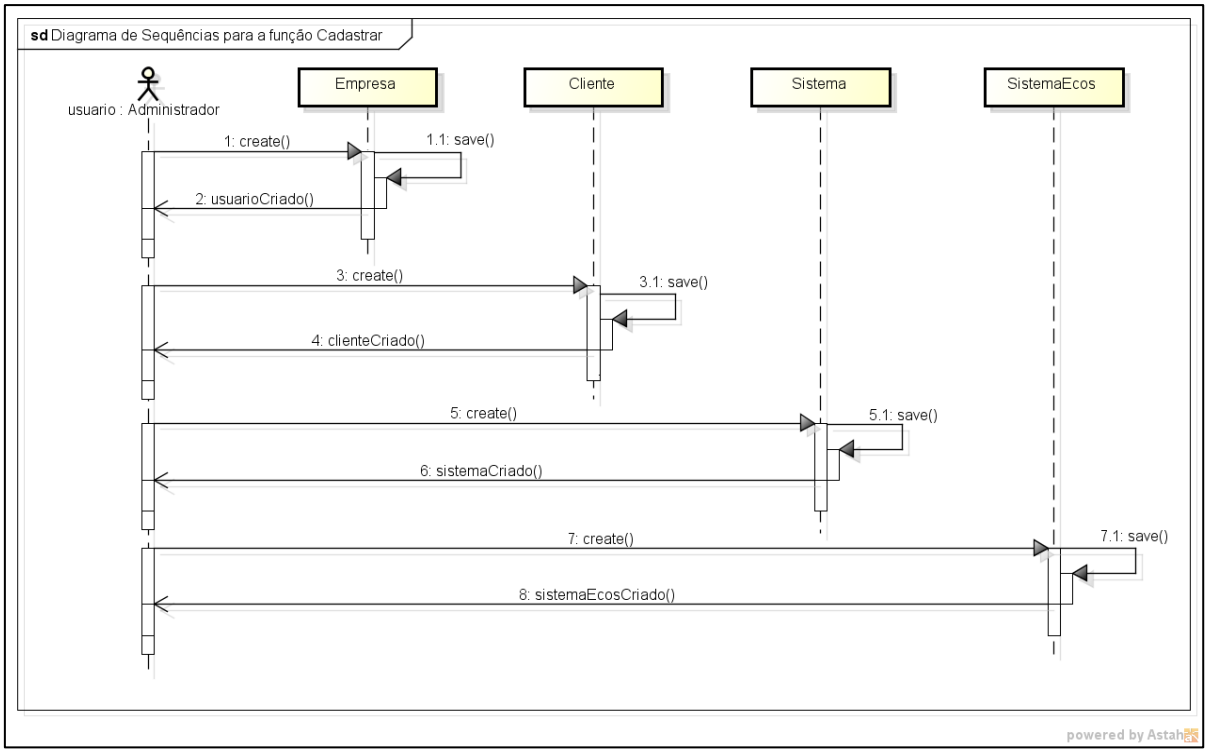
Figura 6 – Diagrama de classes do sistema EMI



Fonte: autoria própria

A Figura 7 mostra o diagrama de sequências para a função cadastrar. Para o cadastro de um cliente é necessário que pelo menos uma empresa já esteja cadastrada, por isso é preciso cadastrar uma empresa inicialmente. O mesmo vale para o cadastro de um sistema: é necessário que estejam cadastrados um cliente e um sistema ECOS.

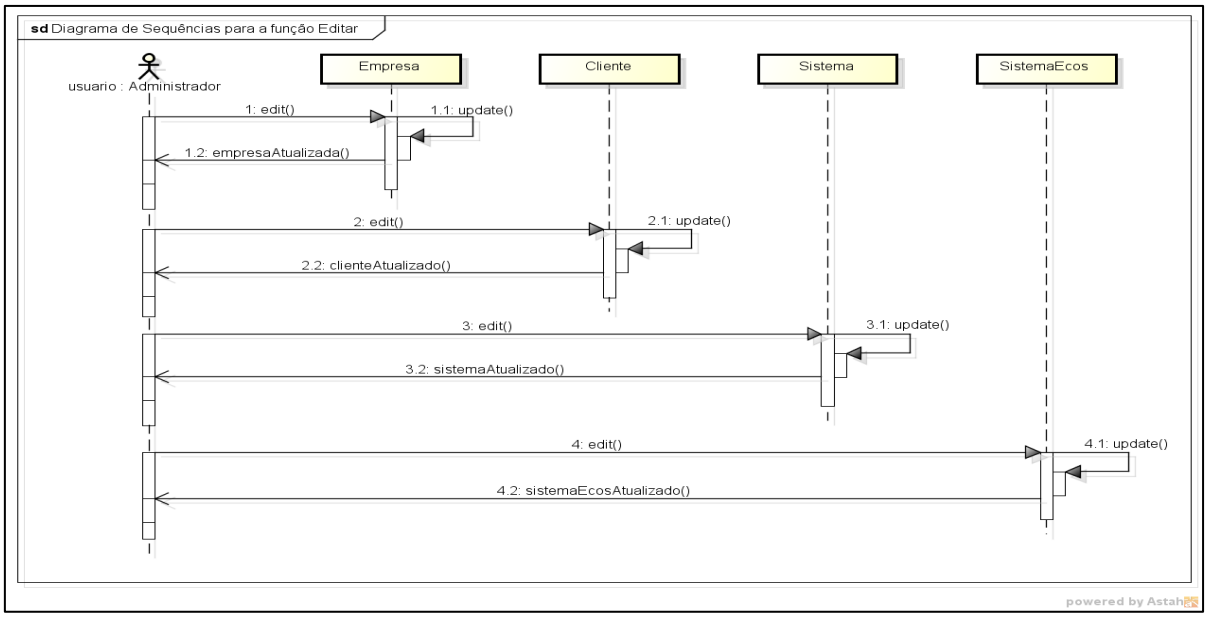
Figura 7 – Diagrama de sequências para a função cadastrar



Fonte: autoria própria

A Figura 8 exibe o diagrama de sequências da função editar. É correto afirmar que só é possível editar qualquer informação, partindo do princípio de que ela foi previamente cadastrada.

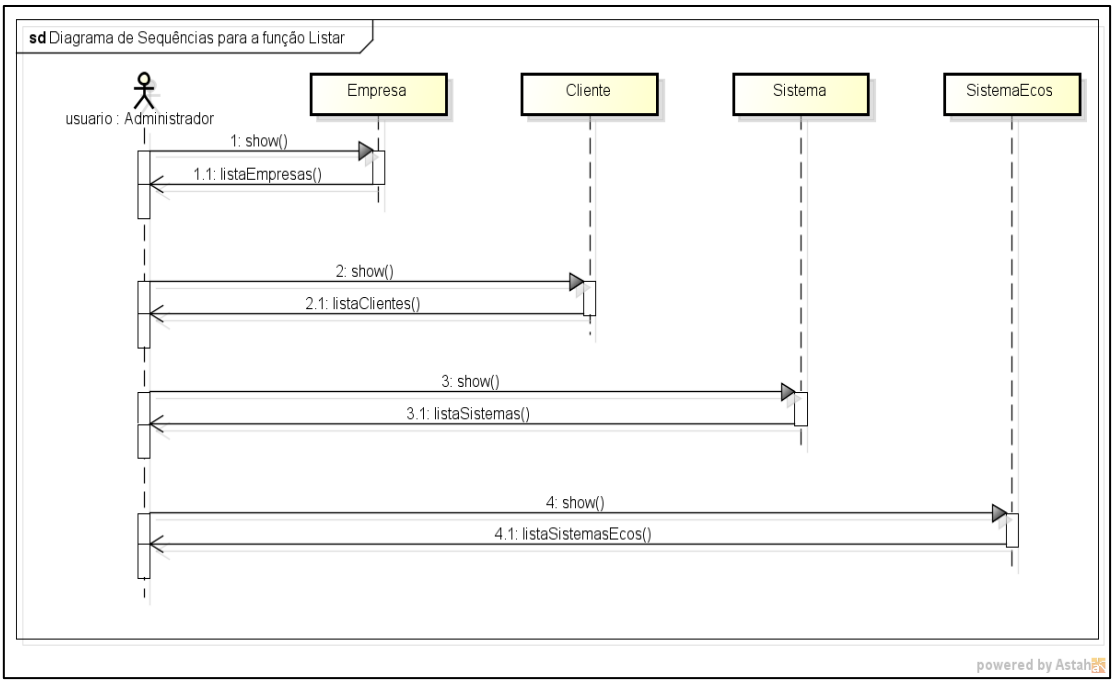
Figura 8 – Diagrama de sequências para a função editar



Fonte: autoria própria

Na Figura 9, é mostrado o diagrama de sequências para a função listar. Quando requisitada, a função listar busca informações do cliente no banco de dados e exibe na tela para o usuário.

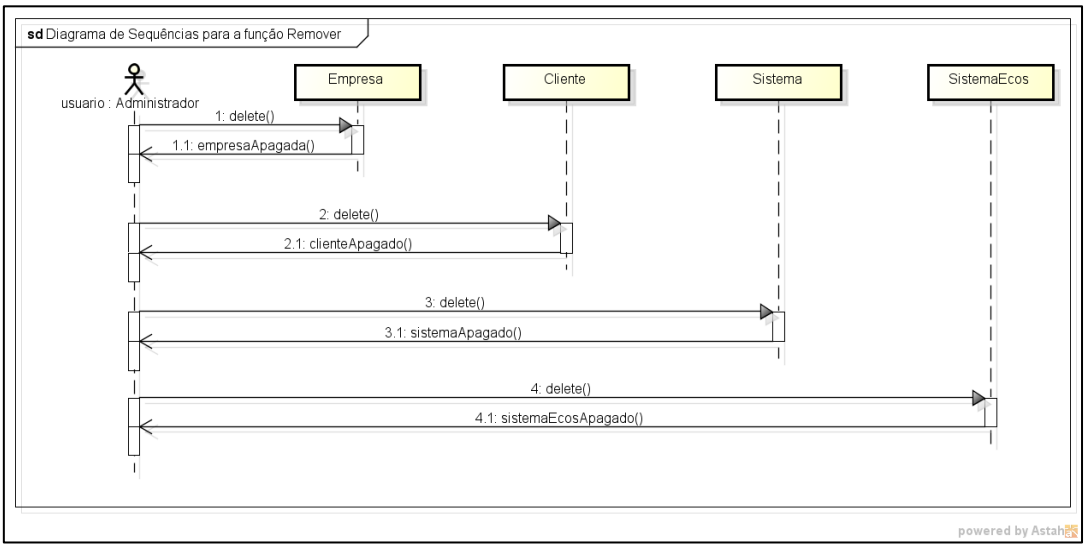
Figura 9 – Diagrama de sequências para a função listar



Fonte: autoria própria

A Figura 10 mostra o diagrama de sequências da função remover. Quando a função remover é requisitada, ela informa ao banco de dados qual informação deve ser apagada.

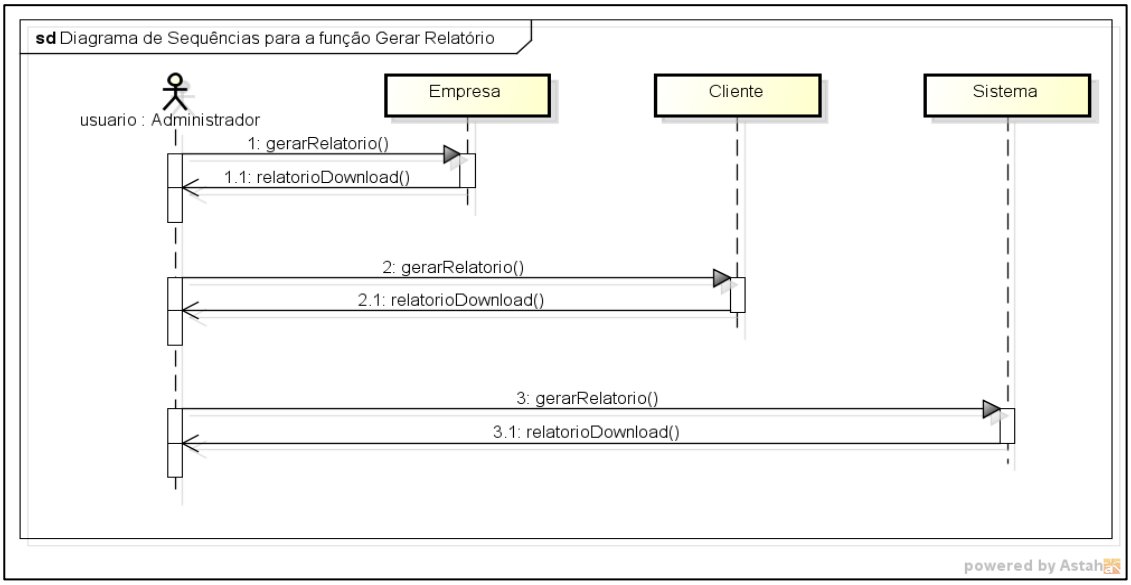
Figura 10 – Diagrama de sequências para a função remover



Fonte: autoria própria

A Figura 11 mostra o diagrama de sequências para a função gerar relatório. O *plugin* responsável por implementar tal função se chama *Jasper*. A função gerar relatório se comporta da seguinte forma: a solicitação é enviada para o *plugin Jasper* que, por sua vez, busca as informações no banco de dados e envia o relatório para o usuário fazer o *download*.

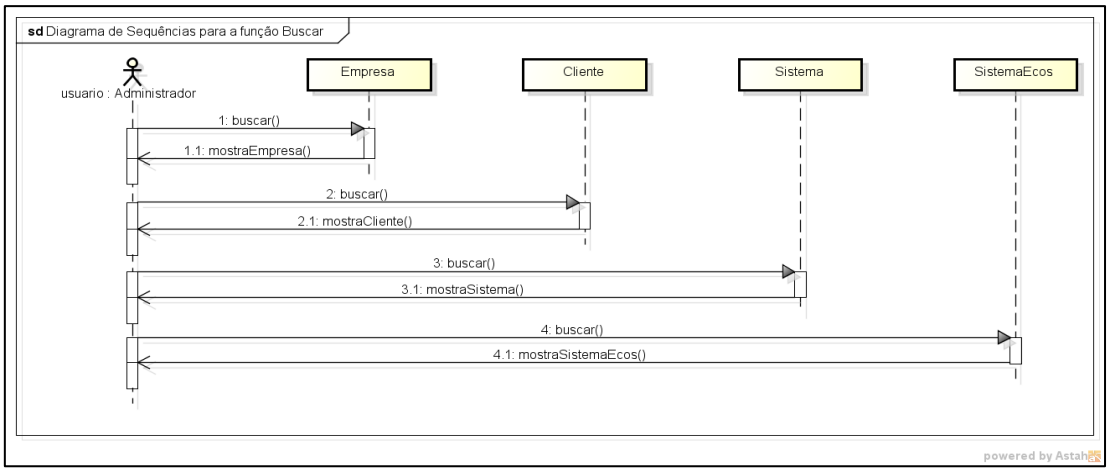
Figura 11 – Diagrama de sequências para a função gerar relatório



Fonte: autoria própria

O diagrama de sequências para a função buscar é mostrado na Figura 12. O *plugin* que implementa a função buscar se chama *Searchable*. Apenas as classes que comportam a função buscar aparecem no diagrama da Figura 12.

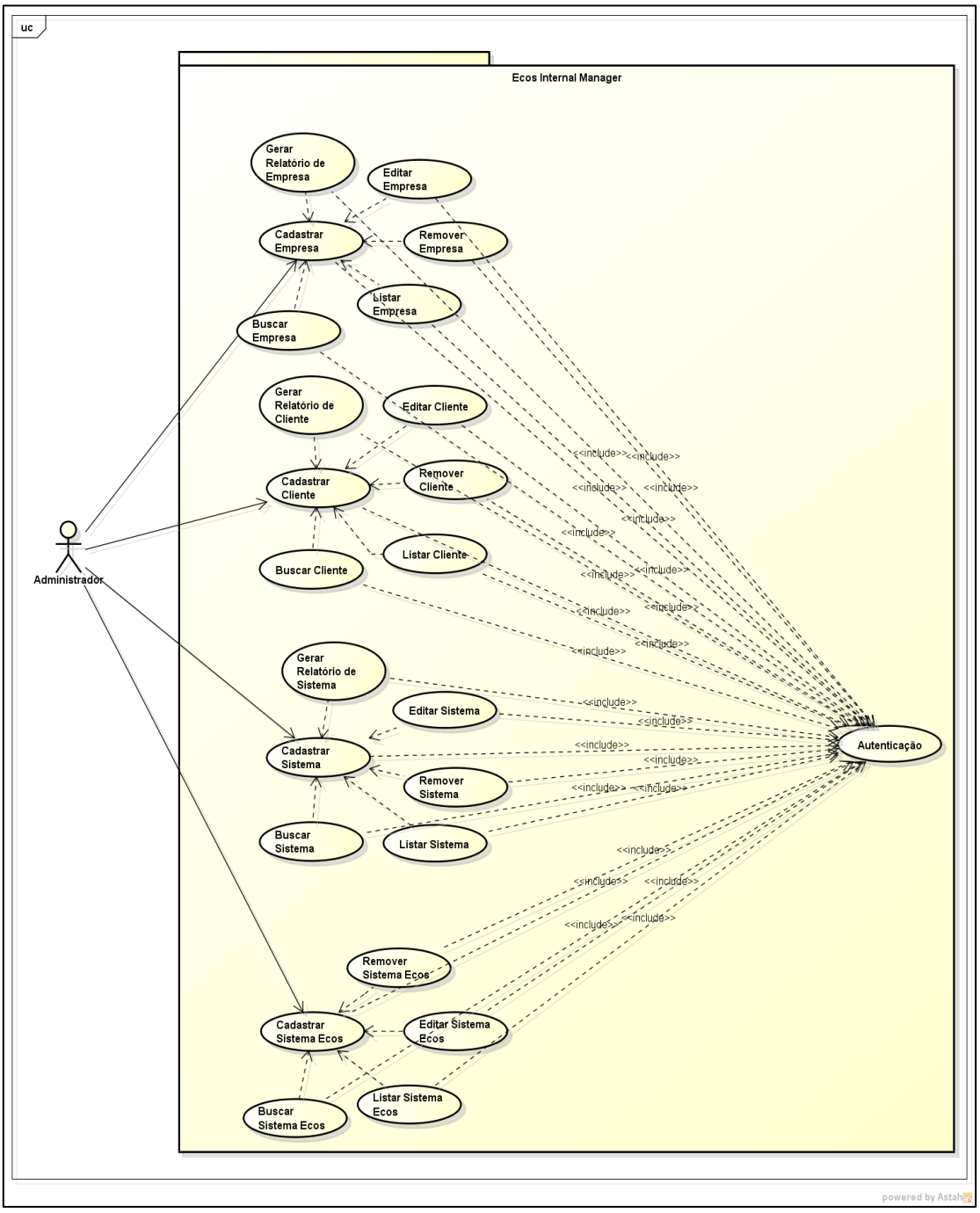
Figura 12 – Diagrama de sequências para a função buscar



Fonte: autoria própria

A Figura 13 mostra o diagrama de casos de uso para o sistema EMI. Como se pode ver, inicialmente, o sistema terá suas funcionalidades acessíveis aos usuários previamente autenticados no sistema. Quando for implementado o controle de acesso por papel de usuário, os administradores terão pleno acesso ao sistema enquanto os usuários com papel limitado terão acesso a apenas algumas funcionalidades.

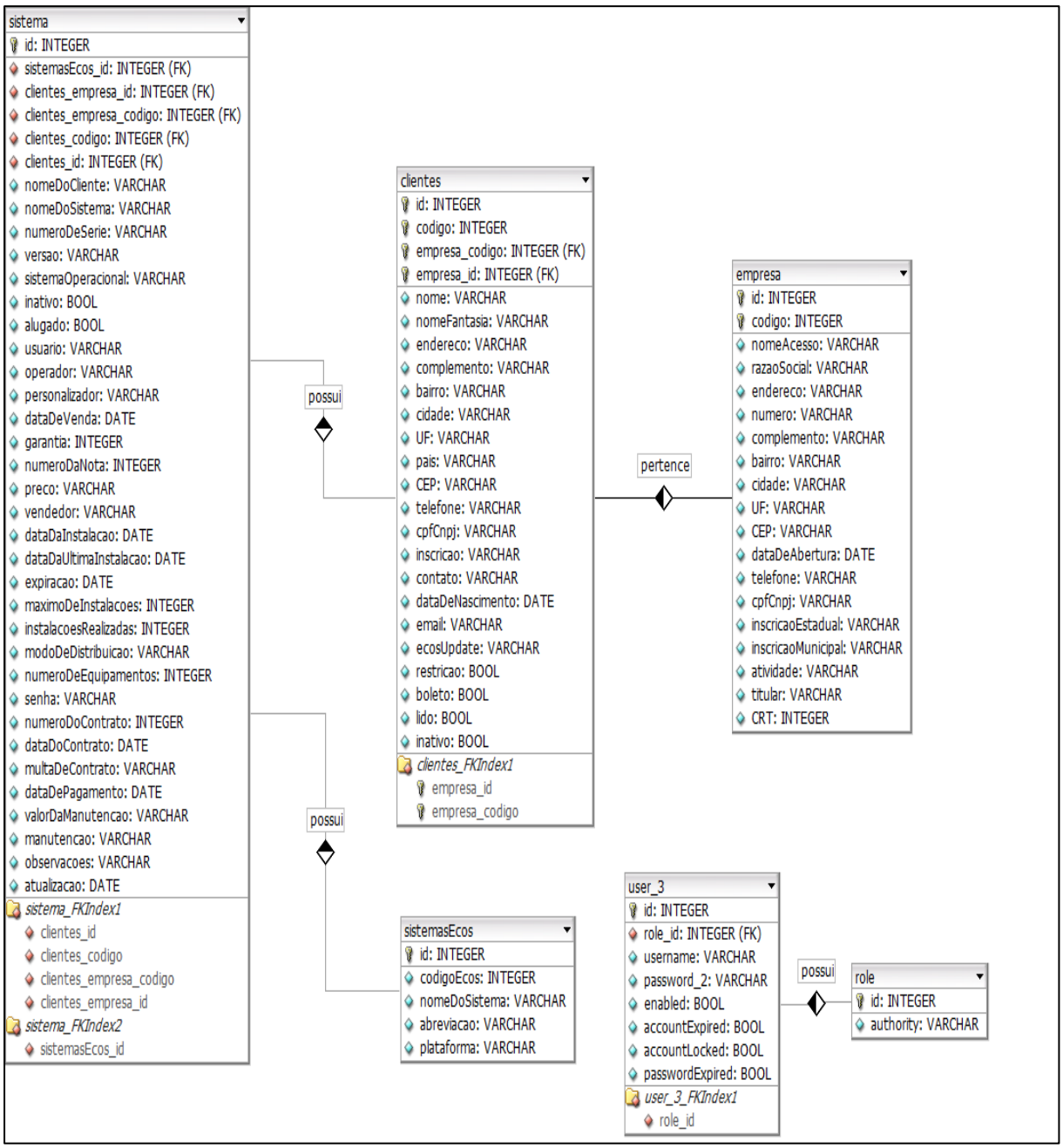
Figura 13 – Diagrama de casos de uso do sistema EMI



Fonte: autoria própria

A Figura 14 mostra o mapeamento objeto-relacional do banco de dados do sistema EMI. Na figura, vemos as chaves estrangeiras de cada tabela e os relacionamentos entre elas.

Figura 14 – Mapeamento objeto-relacional do banco de dados do EMI



Fonte: autoria própria

Os diagramas mostrados nesta seção facilitam o entendimento de como o sistema EMI se estrutura e de como ele foi implementado. Neles, é possível identificar as relações e os processos envolvidos em cada funcionalidade encontrada no sistema.

7.2. DESCRIÇÃO DAS FUNCIONALIDADES DO SISTEMA EMI

Inicialmente, para adquirir acesso ao sistema é necessário o usuário se autenticar. O *plugin* responsável por gerar a tela de *login* e garantir a segurança na autenticação é o *Spring Security*[7], que foi desenvolvido pela empresa Spring. Este *plugin*, além das funcionalidades ditas anteriormente, torna a aplicação segura ao garantir que o desenvolvedor possa implementar o controle de acesso ao sistema através das regras mencionadas na seção que trata do ambiente GGTS encontrada no capítulo 5.

A Figura 15 mostra a tela de *login* do sistema. O desenvolvedor pode customizar a tela de *login*, caso queira.

Figura 15 – Tela de *login* do sistema EMI

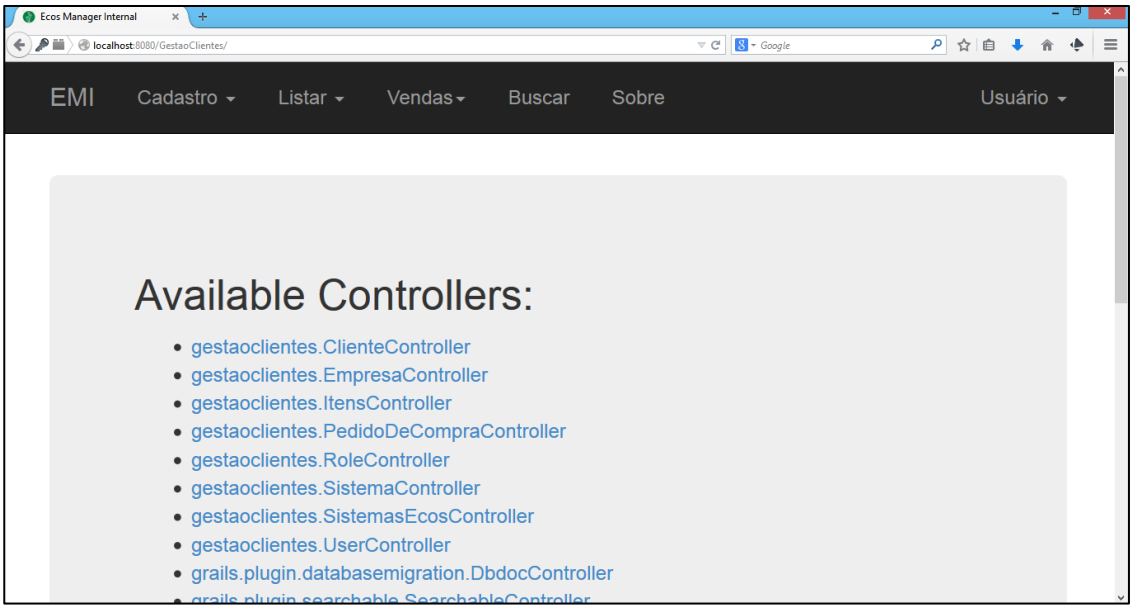
Fonte: autoria própria

A tela inicial, mostrada logo após o usuário se autenticar, consiste em uma barra de menus localizada na parte superior do navegador. As funcionalidades do sistema são acessadas por essa barra. Na tela inicial também são mostrados os controladores de cada classe de domínio e de cada *plugin* instalado no sistema. Alguns controladores são padrões ao ambiente de desenvolvimento, não sendo necessária a utilização deles por parte do usuário do sistema.

[7] http://grails.org/plugin/spring-security-core

A Figura 16 mostra a tela inicial, logo após a autenticação do usuário. Nela, é possível ver a barra com as funcionalidades do sistema e também todos os controles associados àquela aplicação.

Figura 16 – Tela inicial do sistema EMI



Fonte: autoria própria

O usuário para voltar à tela inicial do sistema, seja em qual tela esteja localizado, basta clicar no nome EMI localizado na barra superior.

Para cadastrar alguma empresa, cliente, sistema ou sistema ECOS, basta o usuário clicar no menu “Cadastro” e selecionar a opção que desejar no sub-menu que aparece após o clique no menu.

Após selecionar a opção, o usuário entrará com os dados para o cadastro. Os campos obrigatórios estão identificados com um asterisco ao lado do nome do campo. Depois que o usuário entrar com os dados, para efetivar o cadastro, basta clicar no botão da cor verde identificado como “Criar”.

A Figura 17 mostra a tela de cadastro de um sistema ECOS. O usuário entra com as informações e efetiva o cadastro com o clique no botão “Criar”. O mesmo vale para as outras telas de cadastro.

Figura 17 – Tela de cadastro de um sistema ECOS

Fonte: autoria própria

Para listar, é necessário que o usuário clique no menu “Listar” e selecione a opção no submenu que aparece após o clique. No submenu constará as opções “Empresa”, ‘Cliente”, “Sistema” e “Sistema ECOS”.

A Figura 18 mostra a tela de listagem de clientes. Nela, é possível ver que dá pra inserir um novo cliente (sem precisar clicar no menu “Cadastrar” ➔ “Cliente”) e que também dá pra gerar um relatório contendo as informações de todos os clientes cadastrados.

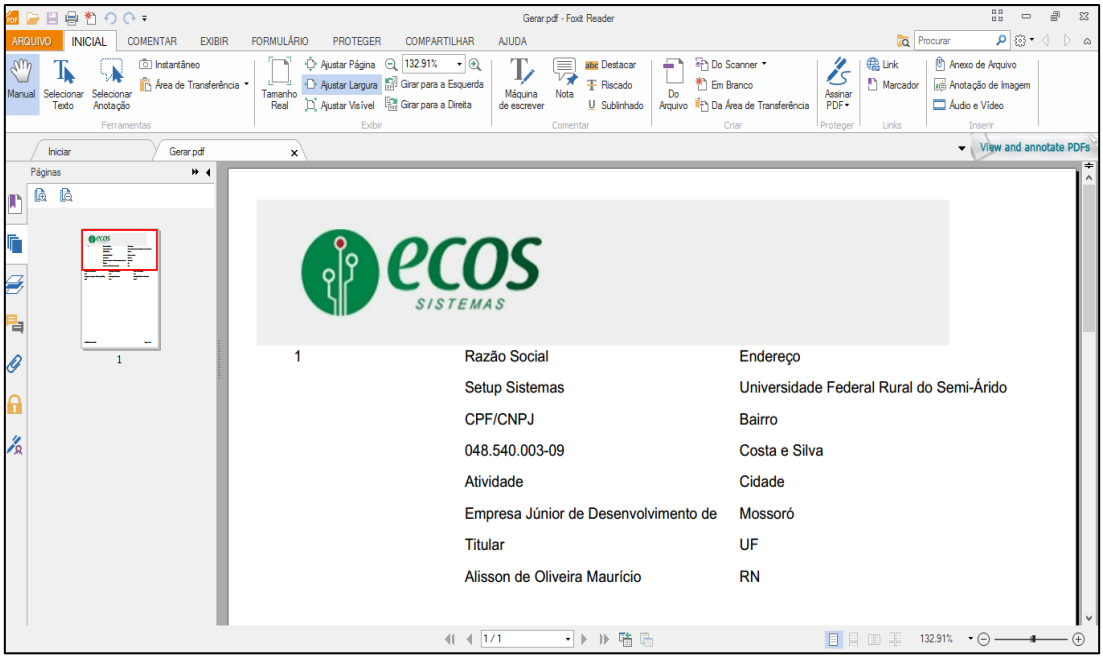
Figura 18 – Tela de listagem de clientes

Fonte: autoria própria

O *plugin* responsável por gerar os relatórios de cada classe de domínio se chama *Jasper*[8], que gera relatórios baseados em modelos elaborados através do programa *iReport*[9]. Tanto o programa quanto o *plugin* foram criados pela empresa *Jaspersoft*. A função gerar relatório foi implementada somente para as classes Empresa, Cliente e Sistema. Os relatórios são gerados para o formato ".pdf".

A Figura 19 mostra um exemplo de relatório gerado pela classe de domínio Empresa. Para gerar o relatório, basta clicar no botão "Gerar Relatório" encontrado na tela de listagem de empresa.

Figura 19 – Exemplo de relatório gerado pela classe de domínio empresa



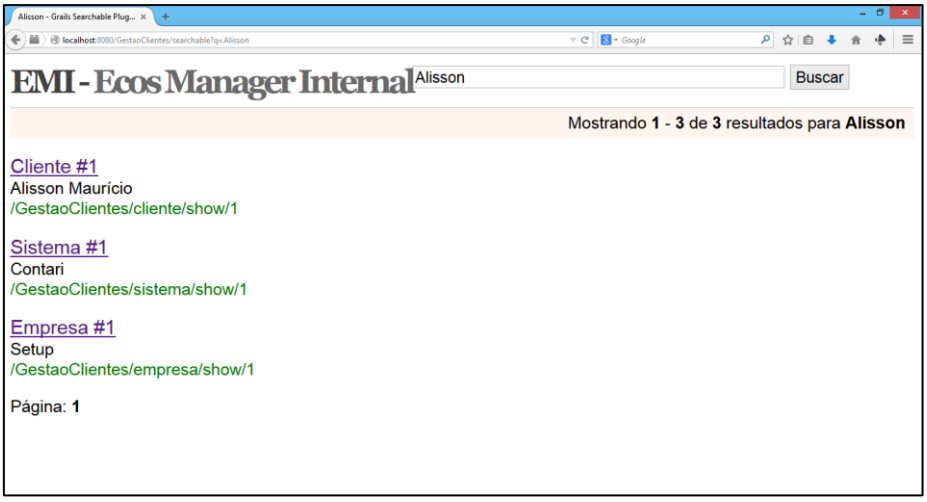
Fonte: autoria própria

A função buscar foi implementada pelo *plugin* chamado *Searchable*[10]. Para fazer uma busca, basta clicar no botão "Buscar" localizado na barra superior. Ao clicar, o usuário será redirecionado para uma página customizada pelo próprio *plugin*. Para realizar a busca, basta digitar o que se deseja buscar na barra e clicar em "Buscar".

A Figura 20 mostra o resultado de uma busca pelo termo "Alisson" que, por sua vez, se estende para todas as classes de domínio do sistema.

[8] http://grails.org/plugin/jasper
[9] http://community.jaspersoft.com/project/ireport-designer
[10] http://grails.org/plugin/searchable

Figura 20 – Tela do resultado da busca pelo termo “Alisson”



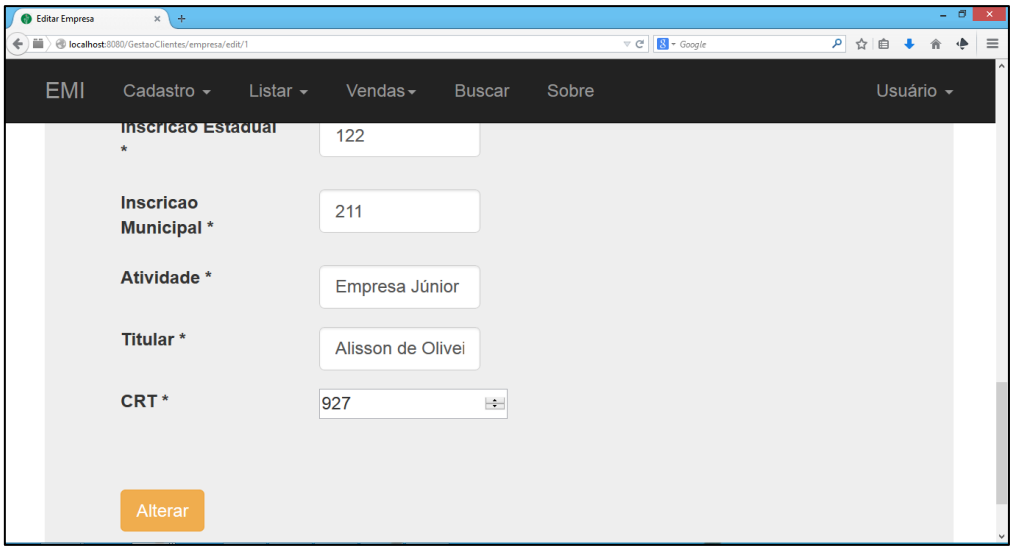
Fonte: autoria própria

Novamente, para o usuário voltar à tela inicial, basta clicar no nome “EMI” localizado na parte superior esquerda do navegador.

Para editar as informações de qualquer empresa, cliente, sistema ou sistema ECOS, basta clicar no menu “Listar”, selecionar a opção que se deseja listar, clicar no código correspondente ao item que se deseja editar e logo após clicar no botão da cor laranja identificado como “Editar”.

A Figura 21 mostra a edição das informações de uma empresa. Após alterar as informações necessárias, para efetivar a confirmação de edição, basta clicar no botão da cor laranja identificado como “Alterar”.

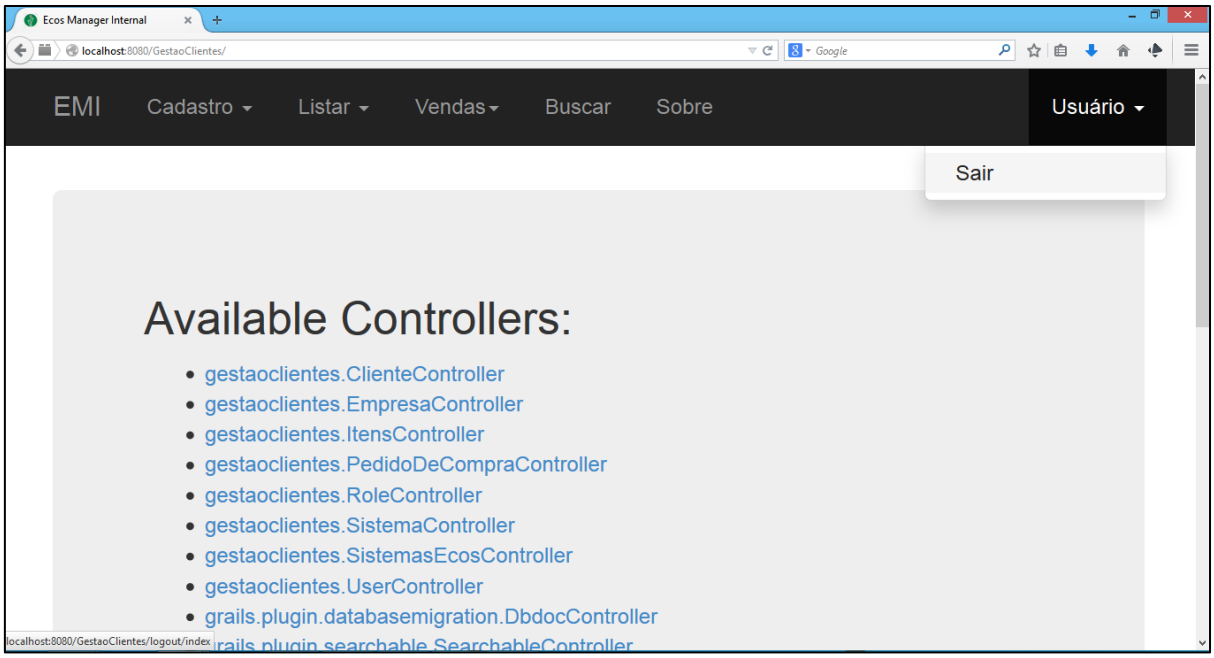
Figura 21 – Tela de edição de uma empresa



Fonte: autoria própria

É mostrado, na Figura 22, como proceder caso queira sair do sistema. Para isso, basta clicar no menu “Usuário” localizado na parte superior direita do navegador e, logo após, clicar em “Sair”. Quando sair, o usuário será redirecionado para a tela de *login*.

Figura 22 – Botão de *logoff* do sistema EMI



Fonte: autoria própria

8. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Inicialmente, foram evidenciados os problemas que o uso de um sistema legado pode acarretar a uma empresa, principalmente quando essa empresa usa esse sistema para gerenciar os seus clientes. Com isso, foi proposta uma solução em forma de um sistema WEB para a gestão de clientes da empresa ECOS Sistemas.

São perceptíveis os avanços que estão sendo feitos no campo de desenvolvimento em aplicações WEB. O *framework Grails* e a linguagem de programação *Groovy* disponibilizam um outro paradigma de programação, que consiste na combinação dos dois. Esta combinação integra diversas técnicas e tecnologias existentes do *framework Grails* com as características dinâmicas da linguagem *Groovy*, resultando em uma estrutura favorável ao desenvolvimento com relação à facilidade de utilização e produtividade.

Depois, foi feito um levantamento teórico sobre as tecnologias utilizadas no desenvolvimento do sistema proposto. As tecnologias consistiam nos *frameworks* utilizados, *Grails* e *Bootstrap*, o primeiro garantindo a alta-produtividade por parte das funcionalidades e o segundo por parte das interfaces gráficas. A linguagem de programação utilizada foi o *Groovy*, que consiste em uma linguagem no padrão *Java*, porém com algumas características que a deixam mais simplificada em relação ao próprio *Java*. O ambiente de desenvolvimento GGTS também foi descrito e exibidos alguns exemplos que ilustravam a facilidade com que ele integrava os *frameworks* e linguagem de programação utilizada, incluindo a simplicidade de integração do ambiente com os *plugins* de terceiros. O banco de dados utilizado foi o *MySQL*.

Logo após, o sistema EMI foi descrito. Durante a descrição foram apresentados alguns diagramas para facilitar o entendimento e, também, foram explanadas as funcionalidades do sistema.

Portanto, o sistema proposto como forma de solução para o problema existente na ECOS deixou de ser um sistema legado, dada a sua migração para uma linguagem de programação mais recente. O problema dos bancos de dados duplos foi resolvido por meio de sua unificação. Com isso, o fato das informações sobre os clientes da empresa estarem desatualizadas foi resolvido, pois o sistema WEB faz com que as informações sejam atualizadas de forma automática.

Podemos destacar as vantagens e as vantagens da utilização do *Ecos Manager Internal* na empresa.

VANTAGENS:

- O idioma está em português do Brasil;
- Interface gráfica amigável;
- Usabilidade intuitiva;
- Rápido acesso às funcionalidades.

DESVANTAGENS:

- Por enquanto, só executa em um servidor local;
- Ainda não foi efetivamente implantado na empresa;
- Ainda não há o controle de acesso ao sistema por papéis de usuários.

Por fim, a maior dificuldade encontrada durante o desenvolvimento do sistema foi o fato de haver pouco conhecimento, por parte do desenvolvedor (autor), tanto com o ambiente de desenvolvimento GGTS quanto com os *frameworks Grails* e *Bootstrap* e a linguagem de programação *Groovy*. Essa dificuldade, com o tempo, foi ultrapassada. Também, não foi possível acompanhar a finalização dos testes previstos e nem a devida implantação do sistema na empresa, por causa do tempo de estágio que foi curto e não contava na programação do mesmo. Portanto, as próximas atividades a serem desenvolvidas em relação ao sistema são a realização dos testes e a implantação efetiva na empresa ECOS Sistemas. Os testes a serem realizados consistem nos testes funcionais (testes de requisitos e de tratamento de erros) e nos testes estruturais (testes de estresse, recuperação e de segurança).

Como trabalhos futuros, sugere-se a realização de todos os testes identificados para a efetivação do uso do sistema. Como o sistema, inicialmente, foi desenvolvido sem diferenciar o usuário-administrador do usuário-limitado, sugere-se que seja feito esse controle de acesso ao sistema com a devida distinção entre usuários. Outra sugestão seria aperfeiçoar as interfaces gráficas de acordo com o perfil da ECOS Sistemas.

REFERÊNCIAS

ALBUQUERQUE, R. R. B. **Desenvolvendo aplicações ágeis e dinâmicas com groovy e grails.** Campina Grande: FACISA, 2011.

FERNANDES, MARCELO. **Conhecendo o Ruby on Rails**. Disponível em: <http://oglobo.globo.com/blogs/tecnologia/posts/2010/11/08/conhecendo-ruby-on-rails-338884.asp>. Acesso em: 15 de julho de 2014.

HTML5DEV. **Bootstrap – Primeiros Passos.** Disponível em: <http://www.html5dev.com.br/2013/01/16/bootstrap-primeiros-passos/>. Acesso em: 11 de julho de 2014.

JUDD, C. M.; NUSAIRAT, J. F.; SHINGLER, J. **Beginning Groovy and Grails: From Novice to Professional**. 1. ed. [S.l.]: Apress, 2008.

LAFORGE, GUILLAUME. **Groovy – A dynamic language for the Java platform.** Disponível em: <http://groovy.codehaus.org/>. Acesso em: 09 de julho de 2014.

LAFORGE, GUILLAUME. **JSR 241: The Groovy Programming Language.** Disponível em: <https://www.jcp.org/en/jsr/detail?id=241>. Acesso em: 09 de julho de 2014.

LEDBROOK, PETER. **Simplified Spring Security with Grails.** Disponível em: <http://spring.io/blog/2010/08/11/simplified-spring-security-with-grails/>. Acesso em: 10 de julho de 2014.

SCRUM. **Scrum.** Disponível em: <http://desenvolvimentoagil.com.br/scrum/>. Acesso em: 22 de julho de 2014.