INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE

SILBERT FIRMINO CARVALHO DE SOUSA

# WEBDIET: UM ESTUDO DE CASO DO USO DE *TEMPLATES* EM *PLAY FRAMEWORK*

NATAL–RN
2015

SILBERT FIRMINO CARVALHO DE SOUSA

# WEBDIET: UM ESTUDO DE CASO DO USO DE *TEMPLATES* EM *PLAY FRAMEWORK*

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Desenvolvimento de *Software* do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Desenvolvimento de *Software*.

Orientador: Leonardo Ataide Minora.

NATAL–RN
2015

W725w Sousa, Silbert Firmino Carvalho de.
Webdiet: Um estudo de caso do uso de templates em play framework / Silbert Firmino Carvalho de Sousa. – 2015.
63 f.

Orientador: Prof. Leonardo Ataide Minora.
Trabalho de Conclusão de Curso (Tecnologia em Análise e desenvolvimento de sistemas) – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, 2015.

1. Quality. 2. Software development. 3. Quick. 4. Efficient. 5. Nutrition Area. I. Minora, Leonardo Ataide. II. Título.

CDU 004.4

Ficha elaborada pela Seção de Informação e Referência da Biblioteca Central Sebastião Fernandes do IFRN.

SILBERT FIRMINO CARVALHO DE SOUSA

**WEBDIET: UM ESTUDO DE CASO DO USO DE *TEMPLATES* EM *PLAY FRAMEWORK***

Trabalho de Conclusão de Curso apresentado apresentada ao Curso Superior de Tecnologia em Desenvolvimento de *Software* do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Desenvolvimento de *Software*.

Trabalho de Conclusão de Curso apresentado e aprovado em 24/3/2015 pela seguinte Banca Examinadora:

BANCA EXAMINADORA

Leonardo Ataide Minora - Presidente
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Alexandre Gomes - Examinador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Leonardo Lucena - Examinador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

## AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me guiado e iluminado no decorrer do curso.
À minha família, em particular aos meus pais, Silvio e Waldenia que apoiaram e incentivaram de todas as formas possíveis os meus estudos.
À minha noiva Laryssa Lopes que sempre esteve do meu lado em todos os momentos, dando força para eu poder enfrentar e vencer os desafios que surgiram nessa etapa.
Ao meu orientador Leonardo Minora, pelos ensinamentos e orientações que tornaram possível a realização deste trabalho.
Aos professores do IFRN que contribuíram com seus ensinamentos durante o curso, tornando possível minha formação profissional.
Aos meus amigos e colegas de curso que juntos trocamos conhecimentos, e sempre mantendo uma relação de companheirismo durante essa longa jornada. OBRIGADO A TODOS!

## RESUMO

A busca pela qualidade sempre será um fator para impulsionar a procura por diferentes modelos ou abordagens para chegar ao objetivo desejado, e isso não é diferente em desenvolvimento de sistemas. O presente trabalho busca apresentar um modelo de desenvolvimento capaz de criar uma aplicação de forma menos cansativa, com rapidez e com eficiência. O sistema descrito foi desenvolvido parcialmente para a atividade da área de nutrição visando mostrar esse poderoso modelo de desenvolvimento de um artefato de *software*.


Palavras-chave: Qualidade. Desenvolvimento de *software*. Rapidez. Eficiência. Área de nutrição.

**ABSTRACT**

The search for quality will always be a factor to boost demand for different models or approaches to reach the desired goal, and this is no different in software development. This paper aims to present a development model able to create an application less tiring, and quicker and more efficient. The described system was developed partly to the Nutrition Area activity aiming to show the application of this powerful model.
Keywords: Quality. Software development. Quick. Efficient. Nutrition.

## LISTA DE ILUSTRAÇÕES

## LISTA DE SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| CPF | Cadastro de Pessoas Físicas |
| CRN | Conselho Nacional de Nutricionistas |
| CSS | *Cascading Style Sheets* |
| HTTP | *Hypertext Transfer Protocol* |
| HTML | *Hypertext Markup Language* |
| IDE | *Integrated Development Environment* |
| IFRN | Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte |
| IMC | Índice de Massa Corporal |
| MVC | *Model, View, Controler* |
| Play | *Play Framework* |
| XML | *Xtensible Markup Language* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Atualmente, os desenvolvedores de sistemas buscam diferentes abordagens para melhorar a qualidade de um *software*, bem como diminuir o tempo e o esforço necessários para produzi-los. As ferramentas hoje utilizadas para essa finalidade são os chamados *frameworks*. Um *framework* basicamente se define como um conjunto de funcionalidades já prontas para uso, como por exemplo: a conexão com o banco de dados, formatação de exibição de campos e entre outras – em outras palavras agilidade no desenvolvimento. A principal característica da utilização dessa abordagem é a promoção de reuso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de *software*.

A busca desenfreada pelo o aumento da qualidade e da produtividade da atividade de desenvolvimento de sistemas é o fator responsável pela a escolha da reutilização de *software*, pois com o reuso de artefatos já desenvolvidos e depurados, é possível reduzir o tempo de desenvolvimento, de testes e as possibilidades de introdução de erros em novos artefatos.

O *framework* utilizado nesse trabalho é o *Play Framework (Play)* para desenvolvimento com a linguagem Java. A principal abordagem é utilizar as características e funcionalidades do sistema de *templates* no *Play* para desenvolver uma aplicação *web* voltada para os profissionais da área de nutrição chamada WebDiet. A definição de *template* nada mais é que um modelo de um documento ou uma apresentação visual a ser seguido. O uso de *template* no *Play* utiliza a abordagem da reutilização, tornando a criação de páginas visuais uma atividade menos desgastante do que o habitual.

## 1. 1 OBJETIVOS

Essa seção mostra os objetivos gerais e os específicos do desenvolvimento do presente trabalho.

### 1.1.1 Objetivos gerais

O objetivo geral deste trabalho é desenvolver um protótipo de uma aplicação para mostrar as características e funcionalidades do sistema de *templates* do *Play Framework*.

### 1.1.2 Objetivos gerais

O objetivo específico de presente trabalho consiste:

a) Desenvolver um protótipo de aplicação *web,* chamada WebDiet - utilizando *Play Framework*;
b) Desenvolver a interface gráfica da aplicação através do uso do *framework* gratuito chamado *Bootstrap*;

## 1.2 METODOLOGIA

A metodologia adotada consistiu em duas fases: fase teórica e fase prática. Na primeira, foi realizado um estudo teórico sobre o *Play Framework* para ter o embasamento necessário para o desenvolvimento deste trabalho. Na segunda fase ocorreu o desenvolvimento de uma aplicação para estudo de caso do mecanismo de *templates* denominada WebDiet. Com isso, foram desenvolvidas algumas

funcionalidades da aplicação para que fosse possível visualizar e descrever o funcionamento das *templates*.

## 1.3 JUSTIFICATIVA

O presente trabalho teve como motivação a importância da função *template* em um ambiente de desenvolvimento de sistemas *web*, especificamente utilizando o *Play Framework*, expondo sua importância para agilidade do processo de programação de um artefato de *software* e contribuindo para o aumento significativo da qualidade e produtividade.

## 1.4 ESTRUTURA DO TRABALHO

Além deste capítulo introdutório, este trabalho de conclusão de curso foi dividido em mais três partes: a primeira descreve o *Play Framework* e seus recursos de desenvolvimento, o *framework Bootstrap* e a integração com o *Play*; a segunda apresenta o estudo de caso; e por fim, a terceira onde são discutidas as conclusões.

# 2 *PLAY*: UM *FRAMEWORK* MVC PARA APLICAÇÕES *WEB*

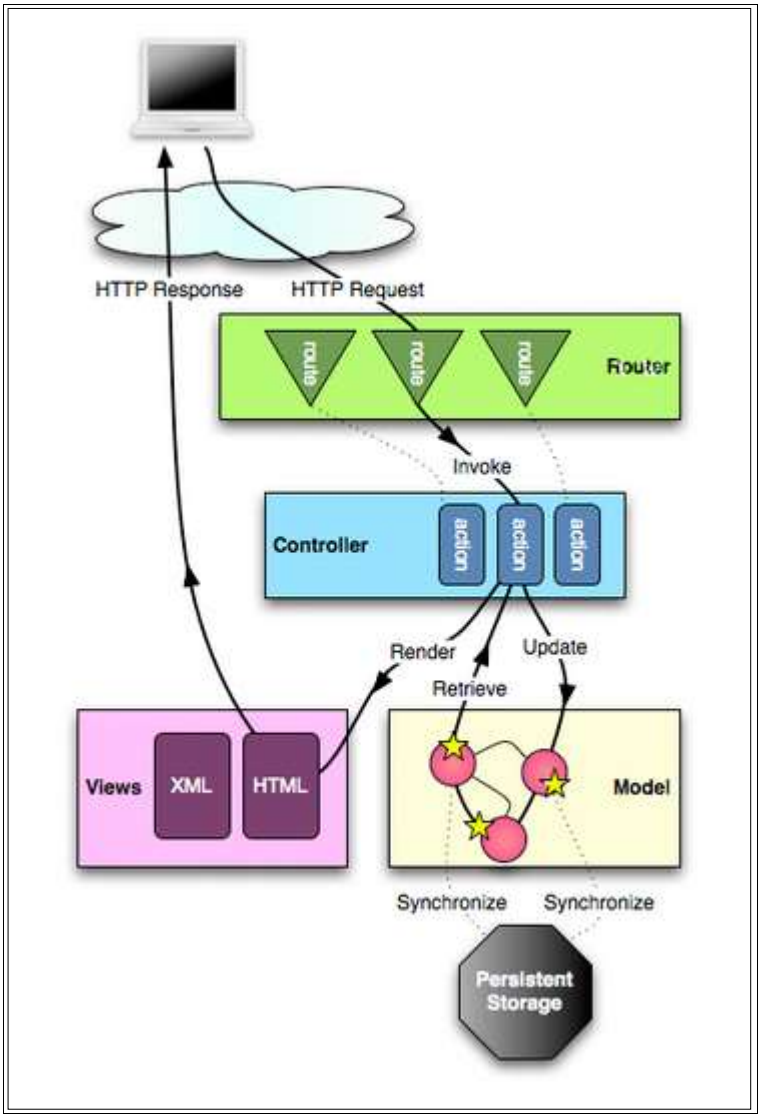
O *Play* surgiu com a necessidade de desenvolvimento de aplicações *web* na linguagem Java de forma ágil e eficiente. Ele é um *framework* multi-plataforma, livre e de código aberto para desenvolvimento de aplicações *web*. Apesar de criado em 2007 por Guilaume Bort, apenas em outubro de 2009 foi publicado sua primeira versão, primeiramente escrito na linguagem Java. Atualmente o *Play* também apresenta a opção de desenvolvimento na linguagem Scala.

O *Play* permite que os desenvolvedores utilizem a *Application Programming Interface* (API) em Java ou Scala para constituírem suas aplicações, sendo o mais aconselhável para iniciantes no *framework* o uso da linguagem Java devido a simplicidade em comparação a linguagem Scala, porém se o desenvolvedor quiser tornar o desenvolvimento ainda mais rápido e produtivo é recomendado utilizar a linguagem Scala devido ao seu código conciso, escalabilidade e o suporte à programação funcional (LEROUX *et al.*, 2012).

O *Play* apresenta como arquitetura o padrão *Model, View, Controler* (MVC). Essa arquitetura é organizada em três camadas: modelo (*Model*), visualização (*View*) e o controlador (*Controller*).

A camada de modelo representa os dados e a lógica de negócios da aplicação, ou seja, uma representação especifica do domínio da informação em que a aplicação opera. A camada *de* visualização é a apresentação na tela, em outras palavras – uma interface com o usuário. Em aplicações *web* a visão geralmente é feito utilizando *Hypertext Markup Language* (HTML) ou *Xtensible Markup Language* (XML). E a camada de controle define a maneira como a interface do usuário reage às entradas do mesmo, ou seja, é o controlador que processa os eventos solicitado pelo usuário. Antes do MVC, os projetos com interface para o usuário tendiam em agrupar todos esses objetos. Nesse sentido, o MVC foi idealizado para aumentar a flexibilidade e a reutilização (GAMMA *et al.*, 2000, p. 20).

Figura 1 - Padrão MVC do *Play Framework*




Fonte: *Play Framework* (2014).

De acordo com a Figura 1, as requisições do usuário são enviadas para o *Controller* através das rotas, que é responsável por mapear qual *Controller* é responsável pela requisição. Depois o *Controller* manipula e valida dados através da camada *Model* e envia para camada *View*, logo em seguida repassa a resposta para o usuário.

O *Play* também apresenta integração com as *Integrated Development Environment* (IDE*)* e editores de textos, tais como: *Eclipse*, *NetBeans*, *IntelliJ*, *TextMate* e *Sublime Text*.

## 2.1 UM PROJETO JAVA NO *PLAY FRAMEWORK*

O *Play* é baseado em comandos, e para utilizá-lo basta baixar uma versão na página de *download* e descompacta-lo. Para iniciar o trabalho de forma eficiente é preciso adicionar o diretório do *Play* no path, ou seja, adicionar um percurso ou caminho de trabalho. Isso permite que só em digitar no *prompt* de comando *Play* tenha-se acesso as funcionalidades do *framework*. O *framework* disponibiliza uma lista de comandos para ajudar o usuário, basta digitar *help*.

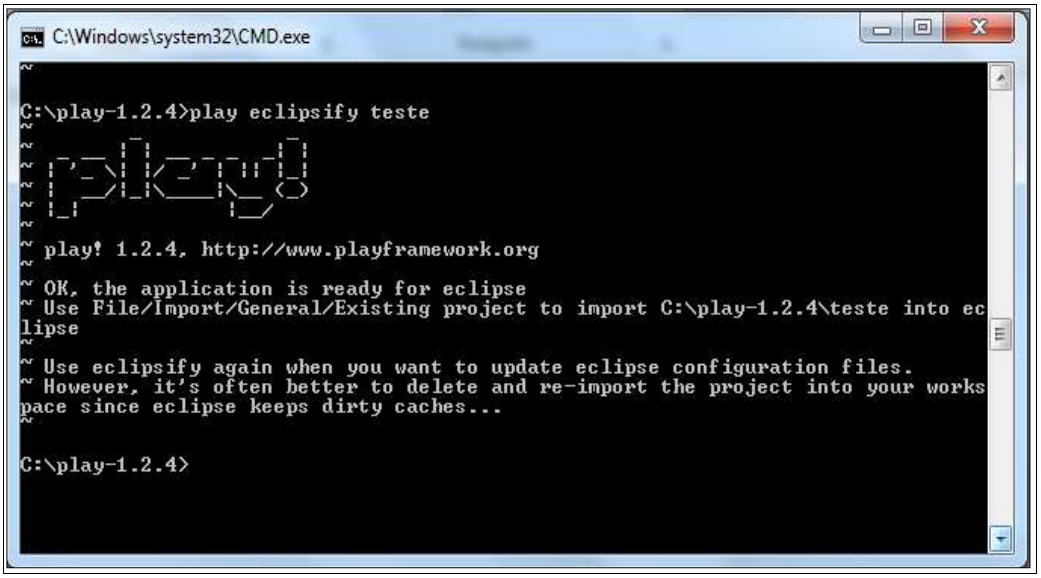
Para criar uma aplicação no *Play* é simples, basta digitar no *prompt* o comando *play new* mais o nome da aplicação. Em seguida aparecerá para confirmar o nome da aplicação caso queira mudar, caso não mude basta apertar *enter*, como mostra a Figura 2 a seguir.

Figura 2 - Criando uma aplicação no *Play Framework*


Fonte: Elaboração própria (2014).

Depois para poder importar essa aplicação para IDE *Eclipse* é ainda necessário preparar esse projeto. Para isso é necessário utilizar o comando *play eclipsify*, com isso o projeto estará pronto para ser importado para a IDE *Eclipse*.

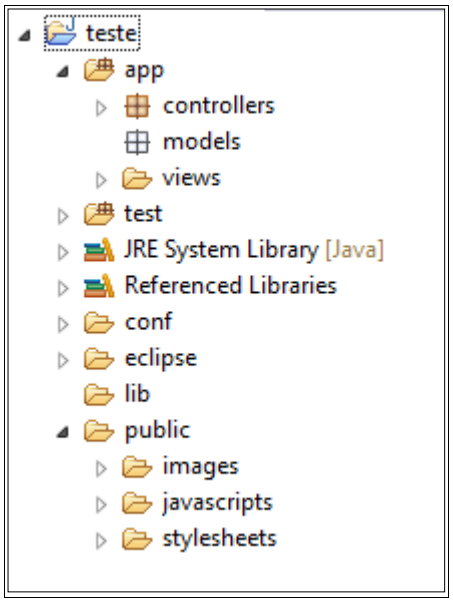
Figura 3 - Preparando o projeto para a IDE *Eclipse.*




Fonte: Elaboração própria (2014).

Em um projeto do *Play* as três camadas (*View, Controller e Model*) são definidas em diretórios cada uma em um pacote Java separado. A Figura 4 mostra a estrutura de diretórios de um projeto no *Play*, na versão 1.2.4, utilizando a *IDE Eclipse.*

Figura 4 - Um projeto no *Play Framework*



Fonte: Elaboração própria (2014).

Cada diretório tem sua função, como veremos a seguir:

a) App/*controllers*: diretório responsável por alocar classes Java que são responsáveis por gerenciar o fluxo da aplicação. A classe *controller* possui uma função processual na aplicação, ela processa uma solicitação *Hypertext Transfer Protocol* (HTTP), trata essa solicitação e envia de volta.
b) App/*models*: responsável por alocar classes Java orientadas a objeto que contém as estruturas e operações em que a aplicação opera os dados.
c) App/*view*: responsável por armazenar páginas HTML e XML de exibição para o usuário. Tudo que ele ver deve ser gerado e exibido por esse diretório.
d) *Test*: diretório de classes testes da aplicação.
e) Conf: contém todos os arquivos de configuração da aplicação.
f) Lib: contém todas as bibliotecas Java necessárias para a aplicação.
g) *Public*: diretório com três subdiretórios: imagens, para armazenar as imagens da aplicação; *Javascripts*, para armazenar eventuais arquivo em *Javacripts* utilizados na aplicação; e o *stylesheets*, para armazenar as folhas de estilos *Cascading Style Sheets* (CSS).

## 2.2 UTILIZANDO *TEMPLATE* NO *PLAY FRAMEWORK*

A criação de telas utilizando HTML ou outra linguagem utilizada para desenvolvimento de páginas *web* sempre foi uma tarefa cansativa e com gasto significativo de tempo. Com o modelo de *template* do *Play* essa tarefa é realizada rapidamente e com pouco esforço. O sistema de *template* presente no *framework* oferece diversas possiblidades para o desenvolvimento seguir no caminho de alta produtividade. O modelo permite a criação de documento formatado com base de texto de forma prática e eficiente. A linguagem *Groovy* é utilizada como linguagem de

expressão, uma linguagem bem similar à do Java, que se integra facilmente com outros códigos e bibliotecas Java.

### 2.2.1 *Template* sintaxe

Em um documento de texto no *Play* podem ser incluídos conteúdos gerados dinamicamente. Esses elementos dinâmicos são escritos na linguagem *Groovy* e são processados durante a execução do documento onde o resultado processado é enviado como parte da resposta HTTP.

#### 2.2.1.1 Expressões

Uma forma de criar elementos dinâmicos é utilizando expressões. As expressões só podem ser utilizadas se estiverem dentro de *${}* (Figura 5).

Figura 5 - Exemplo de expressões

```
1  <ul>
2      <li>Nome: ${cliente.nome}</li>
3      <li>Sobrenome: ${cliente.sobrenome}</li>
4      <li>Email: ${cliente.email}</li>
5      <li>CPF: ${cliente.cpf}</li>
6      <li>Identidade: ${cliente.rg}</li>
7      <li>Telefone Residencial: ${cliente.telefone}</li>
8      <li>Celular: ${cliente.celular}</li>
9  </ul>
```

Fonte: Elaboração própria (2014).

O exemplo ilustra a utilização de expressões em um documento HTML, onde serão exibidas informações de um cliente como: nome, sobrenome, *email*, cpf, rg, telefone e celular.

### 2.2.1.2 Decoradores

Os decoradores são responsáveis por compartilhar *layout* de uma página para vários modelos. A sintaxe utilizada para isso acontecer é *#{extends ”/}*, onde é inserido a página responsável por decorar essa nova página (Figura 6).

Figura 6 - Página decorada

```
1  #{extends 'simpledesign.html' /}
2  #{set title:'A página decorada' /}
3
4  conteudo.
```

Fonte: *Play Framework* (2014).

A Figura 6 mostra que essa página está utilizando elementos de outra página chamada *simpledesign.html*. A Figura 7 apresenta o código da página decoradora *simpledesign.html*.

Figura 7 - Página decoradora (*simpledesign.html*)

```
1  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
2  <head>
3    <title>#{get 'title' /}</title>
4    <link rel="stylesheet" type="text/css" href="@{'/public/stylesheets/main.css'}" />
5  </head>
6  <body>
7    <h1>#{get 'title' /}</h1>
8    #{doLayout /}
9    <div class="footer">Built with the play! framework</div>
10 </body>
11 </html>
```

Fonte: *Play Framework* (2014).

A Figura 7 exibe o elemento *#{doLayout/}*, que é responsável por delimitar onde será exibido o conteúdo das páginas que estendem da *simpledesing.html*, com isso, tornando o desenvolvimento mais rápido e eficiente.

### 2.2.1.3 *Tags*

A *tag* é um fragmento de modelo que pode ser chamado com parâmetros. Se a *tag* tem apenas um parâmetro, por convenção é chamado de *"arg"* e seu nome pode ser omitido (*playframework.org*, 2014), como ilustrado nas figuras 8 e 9.

Figura 8 - Exemplo de *tags*

```
1   #{script 'jquery.js' /}
2   #{script 'jquery.js'}#{/script}
```

Fonte: *Play Framework* (2014).

O exemplo de *tags* exibe duas maneiras que *tags* carregam arquivos *Javascript*. Na Figura 8 na primeira linha, a *tag* é fechada diretamente, quanto que na segunda linha a *tag* é fechada com uma marca final. Ambos estão corretos por possuírem um parâmetro.

As *tags* com mais de um parâmetro não podem ser fechadas diretamente, só com marca final. O exemplo da Figura 9 mostra uma lista *(#{list})* de telefones de um cliente.

Figura 9 - Exemplo de *tags* com dois parâmetros

```
1  <h1>Cliente ${cliente.name}</h1>
2  <ul>
3      #{list items:cliente.telefones, as:telefone }
4          <li>${telefone}</li>
5      #{/list}
6  </ul>
```

Fonte: *Play Framework* (2014).

### 2.2.1.4 *Actions*

As *actions* (ações) são utilizadas com o objetivo de inserir rotas especificas. A sintaxe utilizada é o *@{}* (Figura 10).

Figura 10 - Exemplo de *Actions*

```
1  <h1>Client ${client.name}</h1>
2  <p>
3      <a href="@{Clients.showAccounts(client.id)}">All accounts</a>
4  </p>
5  <hr />
6  <a href="@{Clients.index()}">Back</a>
```

Fonte: *Play Framework* (2014).

A Figura 10 apresenta a utilização de duas *actions*. A primeira *@{Clients.showAccounts(cliente.id)}*, que ao clicar no *link (<a>)* será mostrado todas as contas de um cliente. Funciona da seguinte maneira: ao clicar no *link* a requisição é enviada ao *controller Clients*, que através do seu método *showAccounts(cliente.id)*, responsável por procurar todas as contas de um cliente através de seu *id*, retornará a resposta para o navegador *web* (*view*). A segunda *actions @{Clients.index()}* retornará para uma página definida com o inicial para os clientes.

### 2.2.1.5 Mensagens

As mensagens são utilizadas com o objetivo de internacionalização. Uma mensagem é exibida através de uma palavra ou frase previamente configurada no arquivo *conf/messages* (Figura 11).

Figura 11 - Configurando as mensagens

```
1  clientName=The client name is %s
```

Fonte: *Play Framework* (2014).

Para utilizar é simples, basta chamar em um modelo o nome *clientName* entre aspas simples (Figura 12).

Figura 12 - Utilizando mensagens

```
1  <h1>&{'clientName', client.name}</h1>
```

Fonte: *Play Framework* (2014).

### 2.2.1.6 Comentário

Os comentários são simplesmente notas que podem ser incluídas no código fonte para descrever o que quiser. Os comentários não modificam o programa executado, servem somente para organizar o código fonte. A aplicação dos comentários em um modelo pode ser observada na Figura 13.

Figura 13 - Exemplo de comentário

```
1  *{**** Display the user name ****}*
2
3  <div class="name">
4      ${user.name}
5  </div>
```

Fonte: *Play Framework* (2014).

2.2.1.7 *Scripts*

O *script* é um conjunto mais complexo de expressões, linguagens de programação executadas no interior de programas ou de outras linguagens de programação, não se restringindo a esses ambientes. A sintaxe utilizada para inserir *script* é *%{}.* A Figura 14 ilustra um *script* simples para exibir o nome completo de uma pessoa.

Figura 14 - Exemplo de *scripts*

```
1  %{
2     nomecompleto = cliente.nome.toUpperCase()+' '+cliente.sobrenome;
3  }%
4
5  <h1>Cliente ${nomecompleto}</h1>
```

Fonte: *Play Framework* (2014).

### 2.2.2 *Template* Herança

Uma *template* pode ser utilizada como parte de outra. Isso e possível através da *tag extends*, onde uma página se utilizará de um modelo existente. Na Figura 15 consta uma página HTML chamado *index.html* herdando do modelo *main.html*.

Figura 15 - Página *index.html* utilizando *template*

```
1  #{extends 'main.html' /}
2
3  <h1>Some code</h1>
```

Fonte: *Play Framework* (2014).

A Figura 16 apresenta a *template main.html*, como é chamado a *template* padrão no *Play*, recebendo a *tag doLayout* que é responsável por incluir o conteúdo do modelo que está herdando, no caso *index.html*.

Figura 16 - Página modelo (*main.html*)

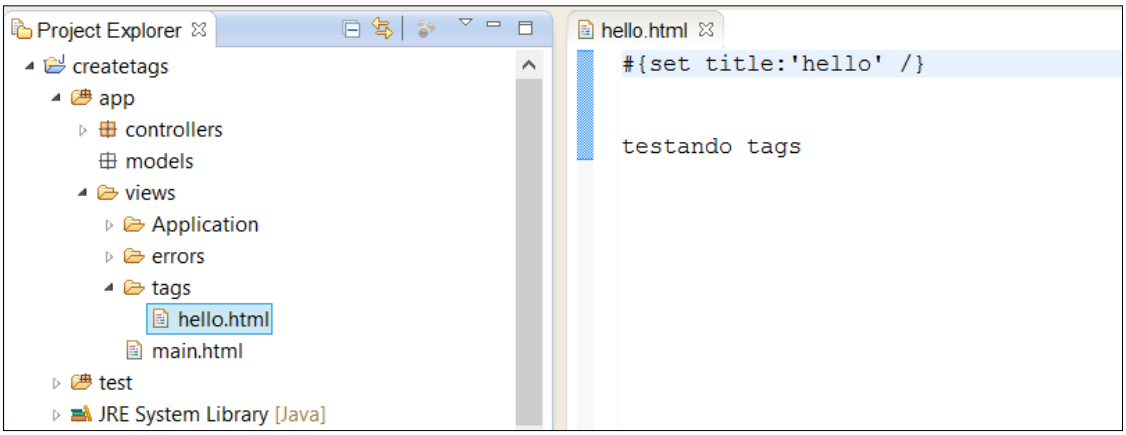
```
1  <h1>Main template</h1>
2
3  <div id="content">
4      #{doLayout /}
5  </div>
```

Fonte: *Play Framework* (2014).

### 2.2.3 Personalizando *tags* em *templates*

No *Play* é possível a criação de *tags* especificas para uma aplicação. A criação de *tags* é realizada no diretório *app/views/tags*, tornando esse arquivo *tag* uma *template*, onde o nome do arquivo é usado como o nome da *tag*.

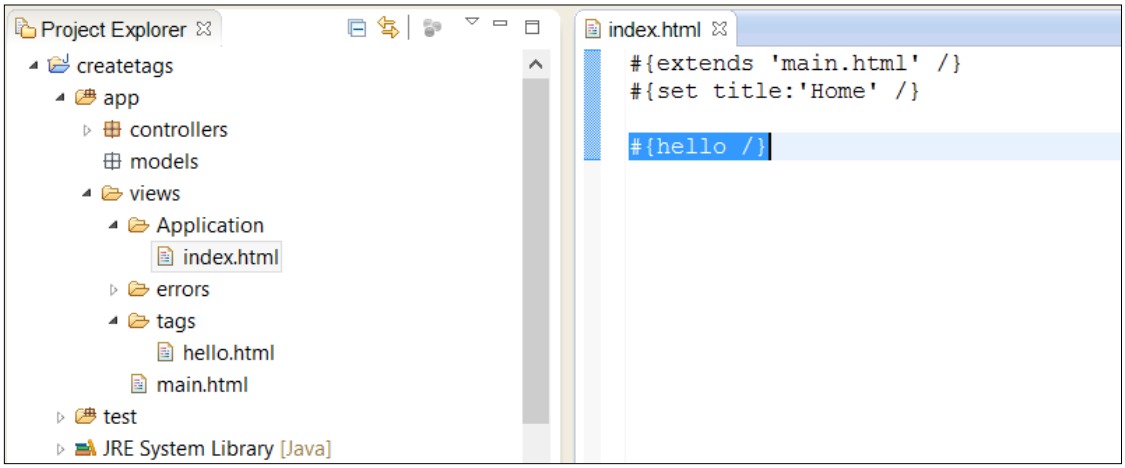
Figura 17 - Criando *tags*




Fonte: Elaboração própria (2014).

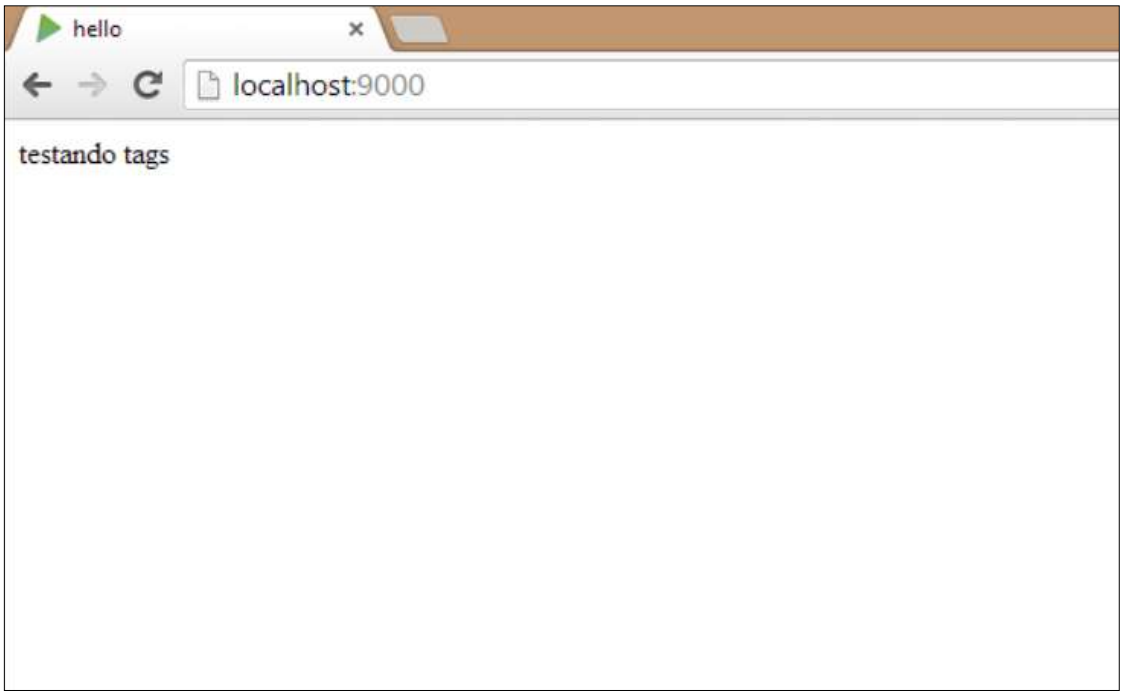
A Figura 17 exibe um projeto do *Play* chamado *createtags*, onde uma *tag* foi criada com o nome de *hello.html*. Para acessar essa *tag*, chamando seu conteúdo, basta utilizar *#{hello/}* sem a necessidade de nenhuma configuração (Figuras 18 e 19).

Figura 18 - Acessando *tag*




Fonte: Elaboração própria (2014).

Figura 19 - Exibindo resultado da utilização de *tags*

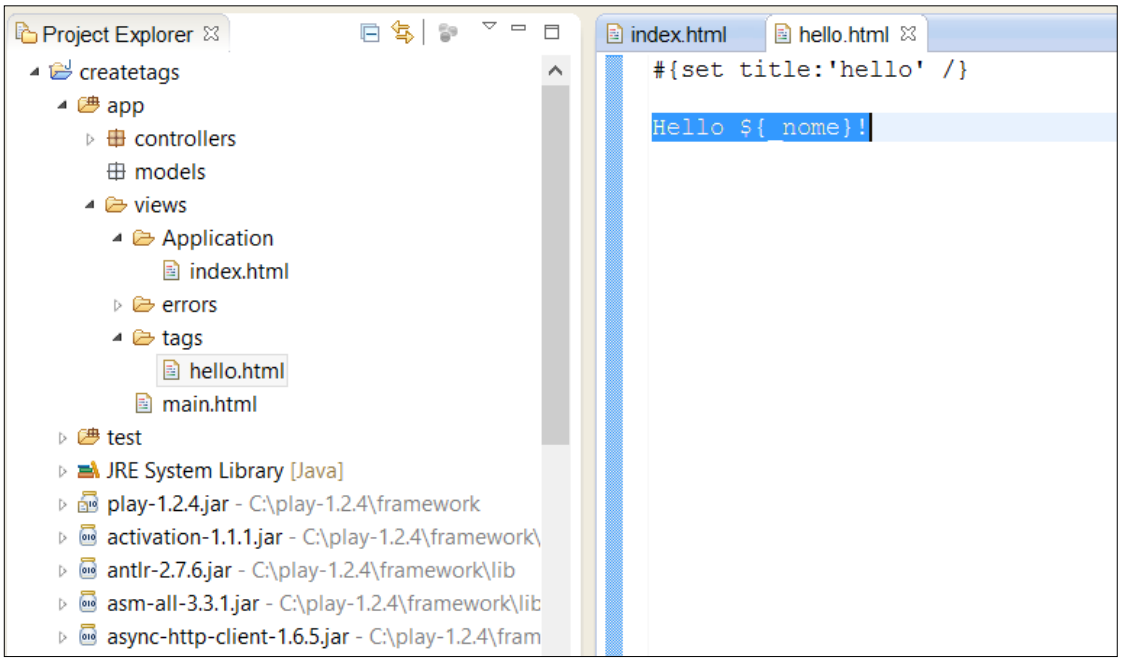



Fonte: Elaboração própria (2014).

### 2.2.3.1 Recuperar parâmetros de uma *tag*

Os parâmetros de uma *tag* na verdade funcionam como variáveis de uma *template*. Por padrão, os nomes das variáveis são construídos com o caractere _ prefixado ao nome do parâmetro, como observado na Figura 20 no mesmo arquivo *hello.html*.

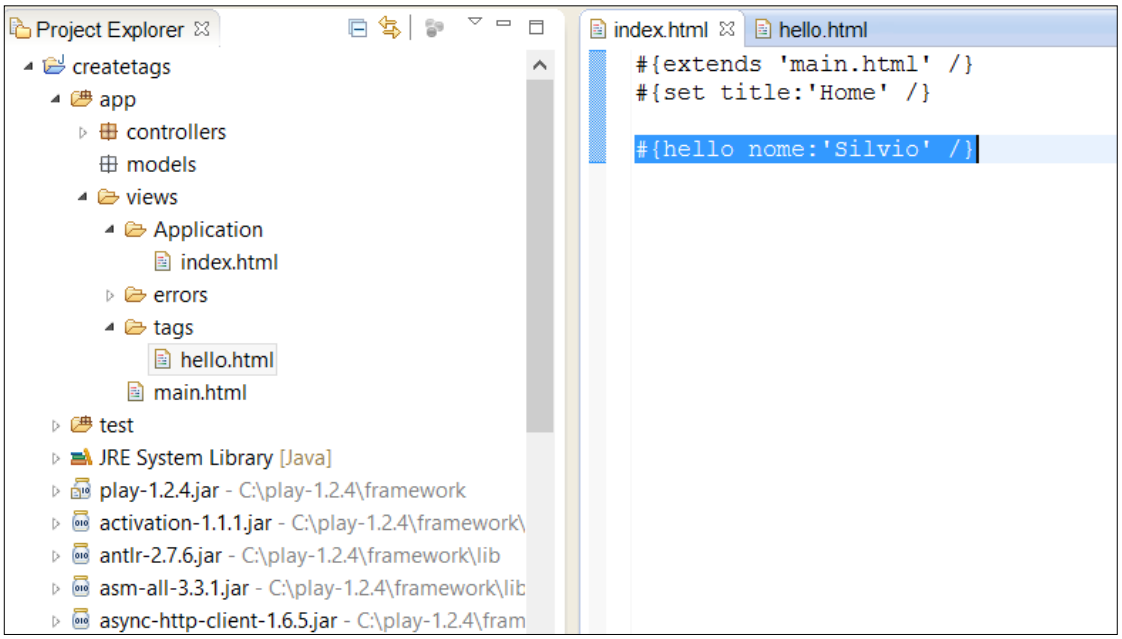
Figura 20 - Recuperando parâmentro de uma *tag* (*hello.html*)




Fonte: Elaboração própria (2014).

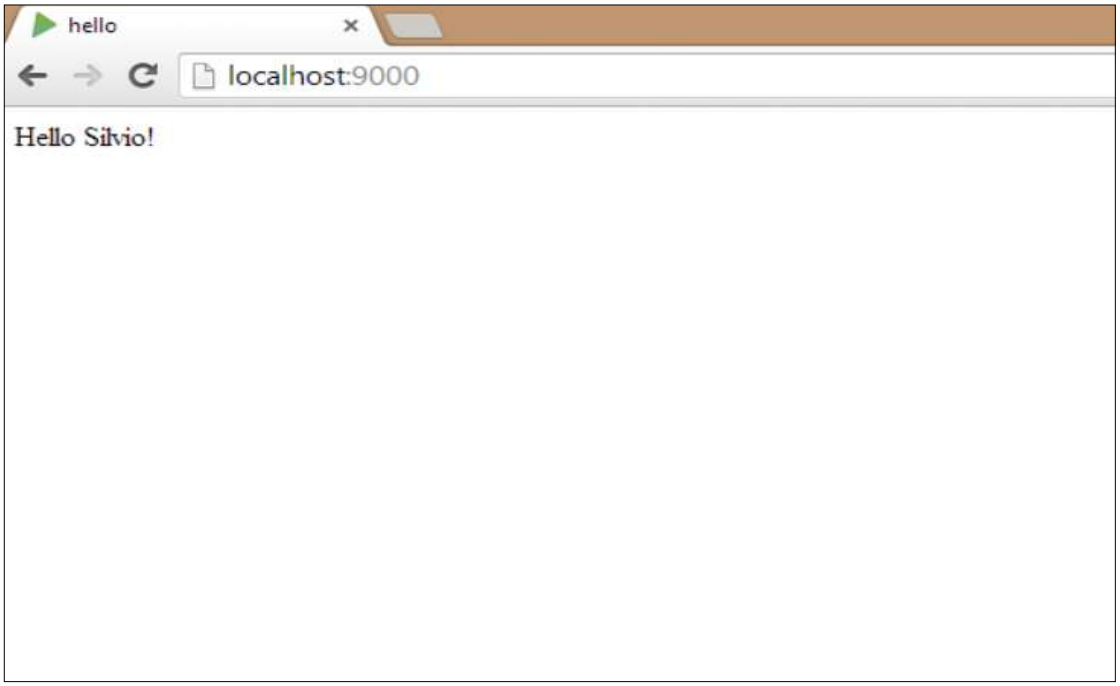
Para passar o parâmetro nome para da *tag hello.html* é utilizado na página de exibição, no caso a *index.html*, a *tag #{hello nome: 'nome'}*. A Figura 21 contém o uso dessa *tag* e a Figura 22 exibe o resultado no navegador.

Figura 21 - Passando o parâmetro nome para a *tag*




Fonte: Elaboração própria (2014).

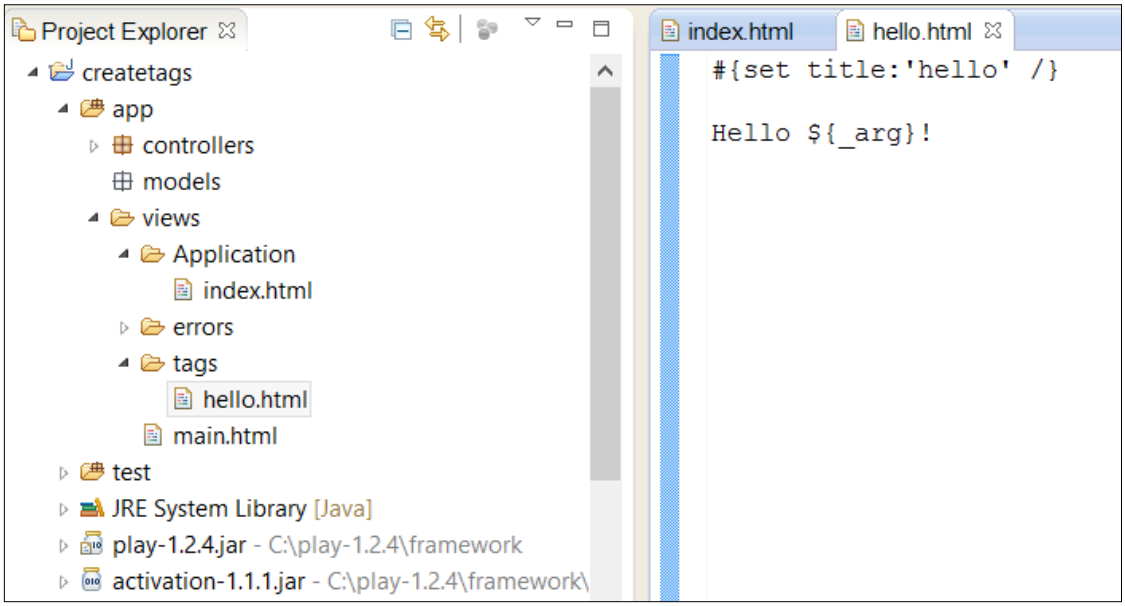
Figura 22 - Exibindo o resultado da recuperação de parâmetro de *tag*




Fonte: Elaboração própria (2014).

Também existe outra maneira de utilizar os parâmetros para *tag*. Se a *tag* apresenta apenas um parâmetro, pode ser usado o fato de que o nome do parâmetro padrão é *arg* e que seu nome está implícito (Figura 23).

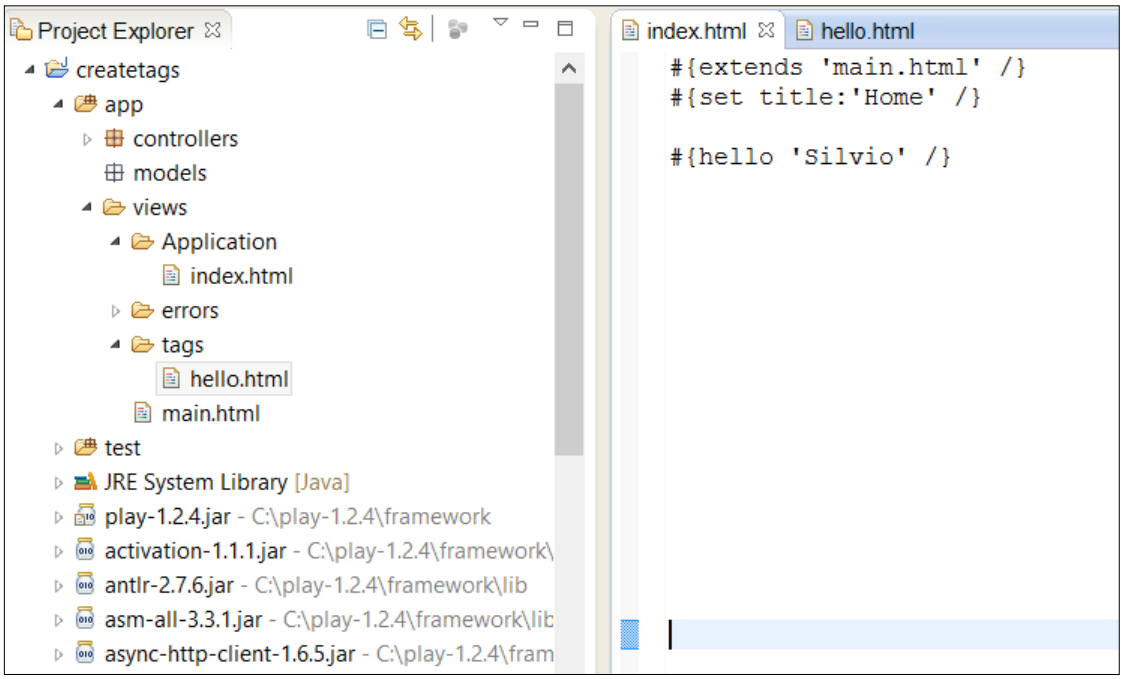
Figura 23 - Recuperando parâmetro padrão *'arg'*




Fonte: Elaboração própria (2014).

O exemplo anterior pode ser facilmente chamado usando *#{hello 'algumnome'}*, como mostra a Figura 24.

Figura 24 - Passando o parâmetro nome implicitamente




Fonte: Elaboração própria (2014).

### 2.2.3.2 Invocar *tag body*

Também é possível incluir um corpo a uma *tag*. Para fazer isso basta utilizar a *tag doBody*, como mostra o exemplo da Figura 25.

Figura 25 - Utilizando a *tag doBody*

```
Hello #{doBody /}!
```

Fonte: *Play Framework* (2014).

E então, é possível passar no nome do corpo a sua *tag*:

Figura 26 - Definindo o corpo da *tag doBody*

```
#{hello}
    Bob
{/hello}
```

Fonte: *Play Framework* (2014)

2.2.3.3 Formato específico de *tags*

A aplicação pode ter diferentes versões de *tags* para diferentes tipos de conteúdo e o *Play* seleciona a *tag* apropriada. Por exemplo, o *Play* usará a *tag app/views/tags/hello.html* quando o formato do pedido for HTML, e *app/views/tags/hello.xml* quando for XML. Seja qual for o tipo de conteúdo, o *Play* vai especificar para a extensão *.tag* caso uma tag de formato especifico não esteja disponível, por exemplo*, app/views/tags/hello.tag.*

2.2.3.4 Personalizando *tags* em Java

No *Play* também é possível criar *tags* utilizando-se de código em Java. Para isso, é necessário criar um método em uma classe que estende *play.templates.FastTags*. Cada método que deseja executar deve ter a assinatura apresentada na Figura 27.

Figura 27 - Assinatura de método em Java para criar *tags*

```
public static void _tagName(Map<?, ?> args, Closure body, PrintWriter out,
    ExecutableTemplate template, int fromLine)
```

Fonte: *Play Framework* (2014).

#### 2.2.3.5 Utilizando *namespace* em *tags*

Para garantir que as *tags* não entrem em conflito, pode-se configurar namespaces utilizando a classe *FastTags* (Figura 28).

Figura 28 - Inserindo *namespace* em *tags*

```
@FastTags.Namespace("my.tags")
public class MyFastTag extends FastTags {
   public static void _hello (Map<?, ?> args, Closure body, PrintWriter out,
      ExecutableTemplate template, int fromLine) {
      ...
   }
}
```

Fonte: *Play Framework* (2014).

Em seguida, nos modelos é feita referência à *tag _hello*, utilizando: *#{my.tags.hello/}*

### 2.2.4 Extensões de objetos Java em *templates*

Quando são utilizados objetos Java em um modelo, novos métodos são adicionados dinamicamente a eles pelo sistema de *templates* do *Play*. Por exemplo, é mostrado uma formatação de números na Figura 29.

Figura 29 - Formatação de números através da extensão de objetos Java

```
<ul>
#{list items:products, as:'product'}
    <li>${product.name}. Price: ${product.price.format('## ###,00')} €</li>
#{/list}
</ul>
```

Fonte: *Play Framework* (2014).

Este exemplo mostra a formatação de números através da classe *Java.lang.Number*, onde foi adicionado pelo sistema de *templates* do *Play* um método para fácil formatação de números. Na documentação do *Play* é possível acessar uma lista que apresenta outras extensões para facilitar o desenvolvimento de um modelo.

#### 2.2.4.1 Criando extensões personalizadas

Em um desenvolvimento de qualquer aplicação, o projeto pode ter necessidades específicas de formatação. Com isso o sistema de *template* do *Play* permite a criação das próprias extensões, basta criar uma classe Java que estenda *play.templates.JavaExtensions.* Essas classes de extensão são automaticamente detectadas pelo *Play*, basta apenas reiniciar a aplicação para funcionar devidamente. Por exemplo – criando um formatador de moeda como representado na Figura 30.

Figura 30 - Criando extensões

```
package ext;

import play.templates.JavaExtensions;

public class CurrencyExtensions extends JavaExtensions {

  public static String ccyAmount(Number number, String currencySymbol) {
     String format = "'"+currencySymbol + "'#####.##";
     return new DecimalFormat(format).format(number);
  }

}
```

Fonte: *Play Framework* (2014).

Cada método de uma classe Java de extensão é estático, e deve retornar uma *string* a ser escrita de volta na página. A Figura 31 exemplifica o retorno para a extensão anterior.

Figura 31 - Utilizando extensão no modelo

```
<em>Price: ${123456.324234.ccyAmount("€")}</em>
```

Fonte: *Play Framework* (2014)

### 2.2.5 Objetos Implícitos em uma *template*

Alguns objetos implícitos estão disponíveis para utilização no sistema de *template* do *Play*. A utilização deles torna o desenvolvimento ágil e eficiente, como o tratamento de erros, retorno de mensagens e entre outros. A lista abaixo mostra alguns desses objetos que podem ser utilizados:

a) *Errors*: validação de erros na aplicação;
b) *Flash*: mantem dados de solicitações HTTP até o próximo pedido;
c) *Messages*: mapeia as mensagens da aplicação;
d) *Out*: gerencia o fluxo de saída;
e) *Params*: extrai parâmetros de dados de solicitações HTTP;
f) *Request*: gerencia solicitação HTTP;
g) *Session*: gerencia a sessão da aplicação.

## 2.3 USANDO A BIBLIOTECA *BOOTSTRAP* NO *PLAY FRAMEWORK*

Nessa seção são apresentadas informações relevantes ao *Bootstrap* e sua utilização no *Play*.

### 2.3.1 Introdução ao *Bootstrap*

O *Bootstrap* é um *framework* de código aberto, criado pela equipe do *twitter*, para facilitar o desenvolvimento de sites e sistemas *web*. Com o *Bootstrap* não será

mais preciso quebrar a cabeça com a formatação do CSS, isso já vem pronto para ser utilizado. O *Bootstrap* também possui vários *plug-ins Javascript* prontos para uso, que com um simples copiar e colar poderá ter em seu site ou na sua aplicação *web* funcionalidades que consumiriam muito tempo para serem criadas do início.

As principais vantagens de utilizar o *Bootstrap*:

a) Possui documentação detalhada e de fácil entendimento;
b) Facilita a criação e edição de *layouts* por manter padrões;
c) Possui uma vasta gama de componentes para o desenvolvimento de qualquer site ou aplicação *web* com interface simples;
d) Funciona em todos os navegadores atuais (Chrome, Safari, Firefox, Internet Explorer, Opera).

Como qualquer ferramenta com o intuito de facilitar determinado trabalho, ela também possui algumas desvantagens, são elas:

a) O projeto deverá seguir os padrões de desenvolvimento do *Bootstrap*;
b) Caso o desenvolvedor não faça ajustes visuais no projeto, o projeto vai parecer com outros que também utilizam o *Bootstrap*.

A Figura 32 exibi como é estruturado o *Bootstrap*, que contém três tipos de arquivos: CSS, *Javascript* e *Fonts*.

Figura 32 - Estrutura do *Bootstrap*

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    └── glyphicons-halflings-regular.woff
```

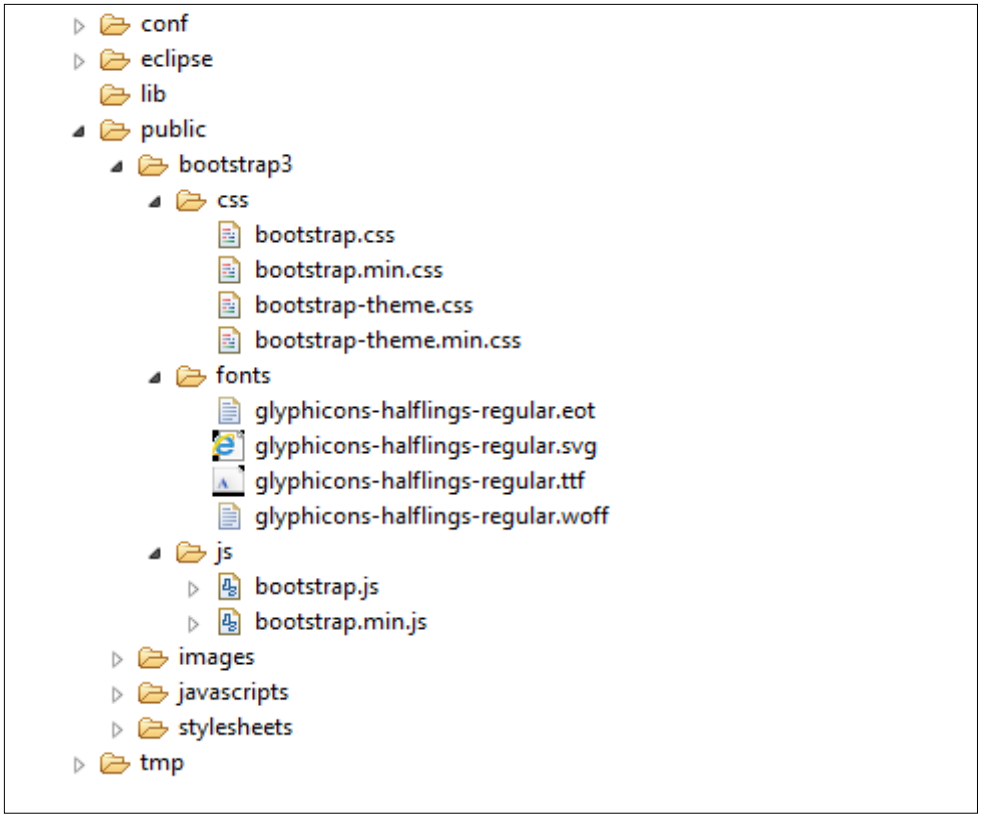
Fonte: *Getbootstrap* (2014).

Os arquivos CSS e *Javascript* vem compilados e prontos para serem utilizados, como também os arquivos da pasta *Fonts*, bastando apenas seguir a documentação do *Bootstrap* para montar um *layout* simples e rápido.

### 2.3.2 Importando o *Bootstrap* no *Play*

O *Bootstrap* foi aplicado no desenvolvimento utilizando o *Play*, para que em conjunto fosse desenvolvida uma aplicação em pouco tempo e com menos esforço. A Figura 33 ilustra onde importar os arquivos do *Bootstrap* no *Play*.

Figura 33 - Importando o *Bootstrap*

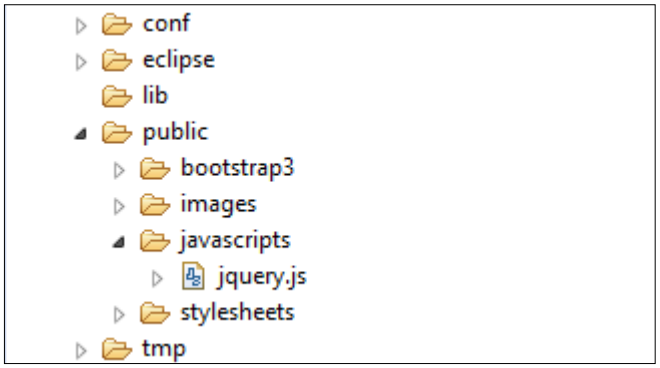



Fonte: Elaboração própria (2014).

A Figura 33 apresenta o esquema de pasta do *Play*, onde os arquivos do *Bootstrap* foram importados para a raiz da pasta *public*, que é a responsável por

organizar as folhas de estilo, imagens e os arquivos *Javascript*. Para funcionamento de todos os *plugins Javascript* é fundamental que o *JQuery* seja incluído no projeto (Figura 34).

Figura 34 - Incluindo o *jquery* ao projeto




Fonte: Elaboração própria (2014).

### 2.3.3 *Template* Básica do *Bootstrap* no *Play*

O desenvolvimento de uma aplicação com *Bootstrap* é muito simples, a Figura 35 exibe uma *template* básica no *Play* preparada para o uso do *Bootstrap*.

Figura 35 - *Template* básica com *Boostrap* no *Play Framework*

```
<!DOCTYPE html>

<html>
    <head>
        <title>Bootstrap Teste</title>
        <meta charset="${_response_encoding}">


        <!-- Le HTML5 shim, for IE6-8 support of HTML5 elements -->
        <!--[if lt IE 9]>
            <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
        <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/bootstrap.min.css'}">

        #{get 'moreStyles' /}

        <script src="@{'/public/javascripts/jquery.js'}" type="text/javascript" charset="UTF-8"></script>
        <script src="@{'/public/bootstrap3/js/bootstrap.min.js'}" type="text/javascript" charset="UTF-8"></script>

        #{get 'moreScripts' /}

    </head>
    <body>

        <h1>Hello Word!</h1>

    </body>
</html>
```

Fonte: Elaboração própria (2014).

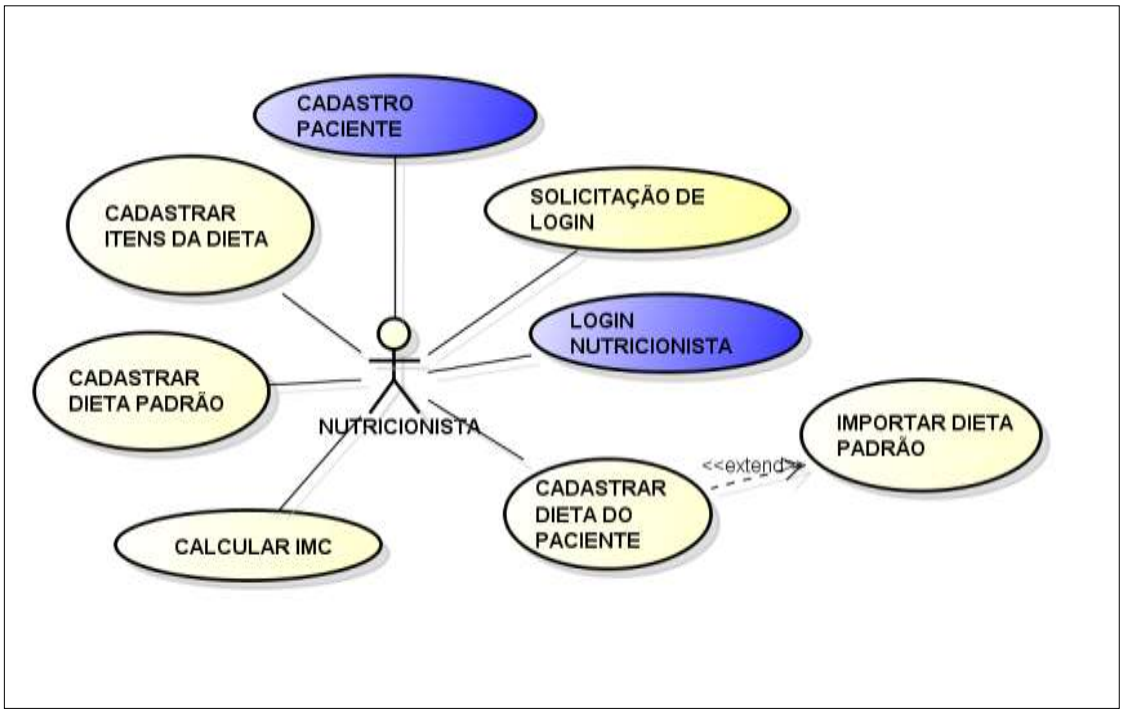
O arquivo é um HTML básico com as devidas inclusões das bibliotecas responsáveis para o funcionamento do *Bootstrap* (*bootstrap.min.css*, *jquery.js* e *boostrap.min.js*) (Figura 35).

# 3 WEBDIET: UM ESTUDO DE CASO DO USO DE *TEMPLATES* EM *PLAY FRAMEWORK*

O WebDiet é um sistema voltado para área de nutrição, em específico para o profissional nutricionista, com o objetivo de facilitar as tarefas básicas da profissão, tais como: administrar dietas, realizar cálculos específicos da nutrição e gerenciar cadastro de pacientes.

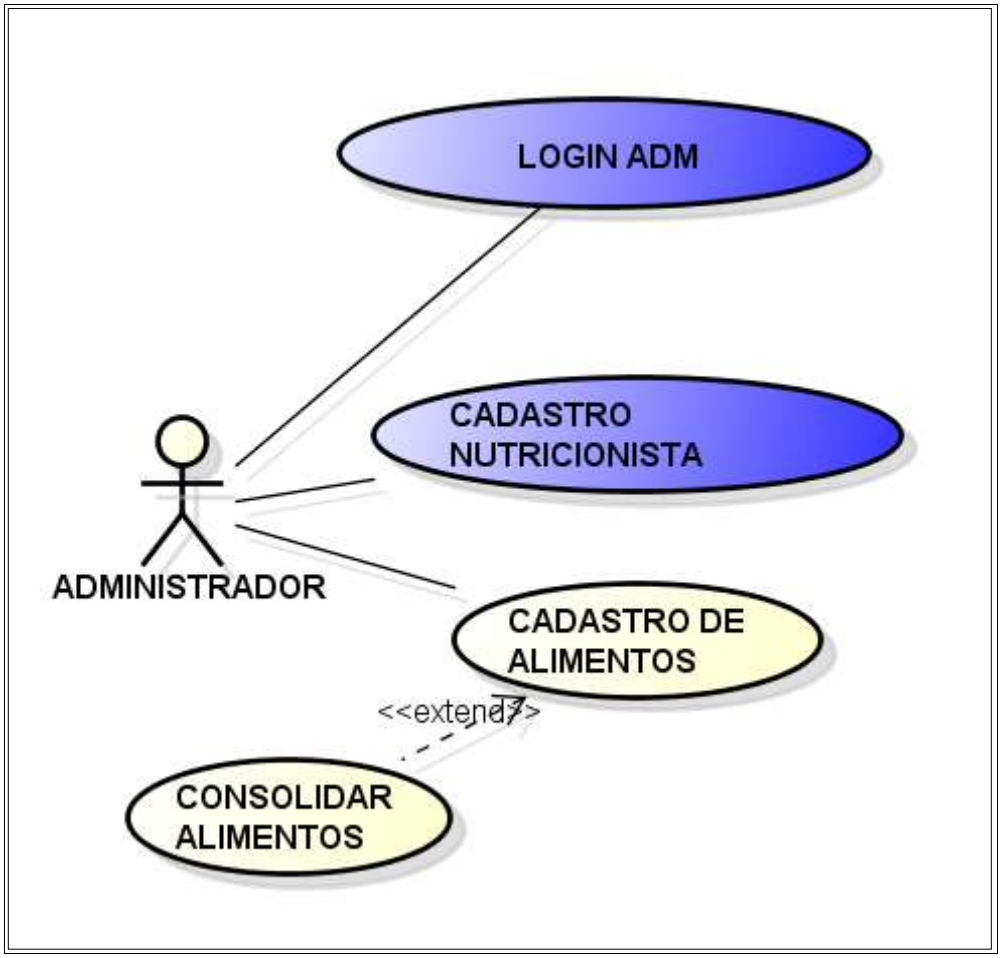
São três os papeis de usuário do sistema: nutricionista, administrador e paciente. A Figura 36 ilustra os casos de uso do ator nutricionista, a Figura 37 mostra os casos de uso do ator administrador, enquanto a Figura 38 exibe os casos de uso do paciente. O Quadro 1 resume esses casos de usos.

Figura 36 - Casos de uso do nutricionista



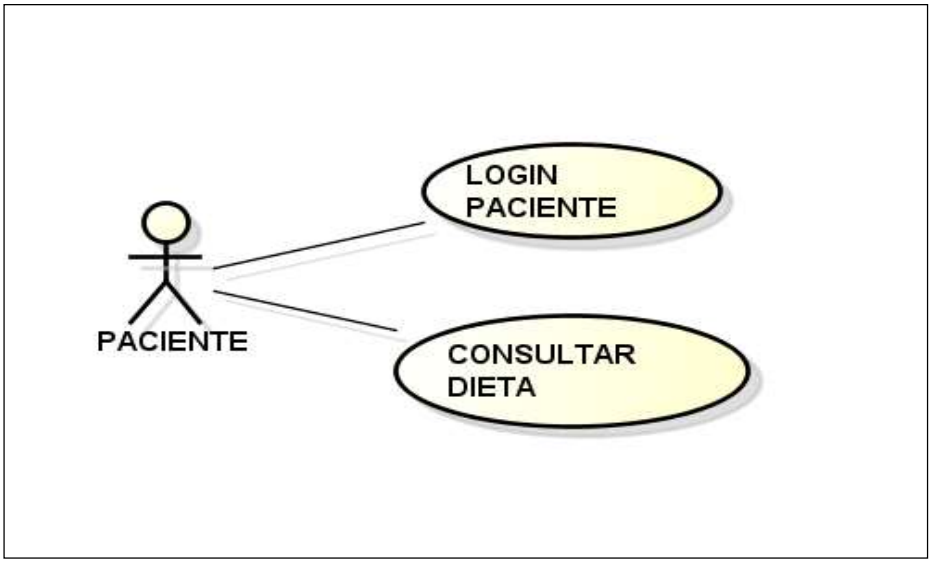

Fonte: Elaboração própria (2014).

Figura 37 - Casos de uso do administrador




Fonte: Elaboração própria (2014).

Figura 38 - Casos de uso do paciente




Fonte: Elaboração própria (2014).

Quadro 1 - Casos de uso do sistema Webdiet

| **Casos de uso** | **Ator** | **Descrição** |
|---|---|---|
| Solicitação de *login* | Nutricionista | O nutricionista solicita autorização para ingressar no sistema. |
| *Login* nutricionista | Nutricionista | Permite ao nutricionista acesso ao sistema. |
| Cadastro paciente | Nutricionista | Gerencia informações relacionadas ao paciente. |
| Cadastrar dieta padrão | Nutricionista | Cadastro de dietas modelos no sistema. |
| Cadastrar itens da dieta | Nutricionista | Cadastro de alimentos na dieta. |
| Calcular IMC | Nutricionista | Permite ao nutricionista calcular o índice de massa corporal (IMC) do paciente. |
| Cadastrar dieta do paciente | Nutricionista | Cadastro de dietas do paciente no sistema. |
| Importar dieta padrão | Nutricionista | O nutricionista importa um modelo de uma dieta já cadastrada. |
| *Login* adm | Administrador | Permite ao administrador acesso ao sistema. |
| Cadastro nutricionista | Administrador | Permite ao administrador gerenciar o nutricionista no sistema. |
| Cadastro de alimentos | Administrador | Cadastro de alimentos no sistema. |
| Consolidar alimentos | Administrador | Consolidar alimentos na dieta. |
| *Login* paciente | Paciente | Permite ao paciente acesso ao sistema. |
| Consultar dieta | Paciente | Permite ao paciente consultar sua dieta. |

Fonte: Elaboração própria (2014).

## 3.2 TECNOLOGIAS UTILIZADAS

O WebDiet é um sistema *web* desenvolvido na linguagem Java através do *Play Framework* 1.2.4, com o *Bootstrap* 3 e o banco de dados relacional *MySQL* 5.

Para desenvolvimento dos diagramas foi utilizada a ferramenta *Astah comunity* 6.8 e para a edição dos códigos-fontes o Eclipse *kepler*.

## 3.3 DELIMITAÇÃO DO TRABALHO

O presente trabalho apresenta um protótipo para demonstrar o uso de *templates* do *Play* com páginas em *Bootstrap*, e como tal não apresentará todas as funcionalidades da aplicação WebDiet. Os casos de uso com cor de preenchimento preto, das Figuras 36 e 37, serão utilizados para demostração: crud paciente, login nutricionista, crud nutricionista e *login* administrador.

## 3.4 ARQUITETURA DAS PÁGINAS DO WEBDIET

O sistema WebDiet foi desenvolvido utilizando o *template* do *Play* como também o *Bootstrap*. A tela inicial da aplicação em desenvolvimento pode ser observada na figura 39.

Figura 39 - Página inicial em desenvolvimento da aplicação WebDiet


Fonte: Elaboração própria (2014).

Foram criados alguns modelos no projeto WebDiet como:

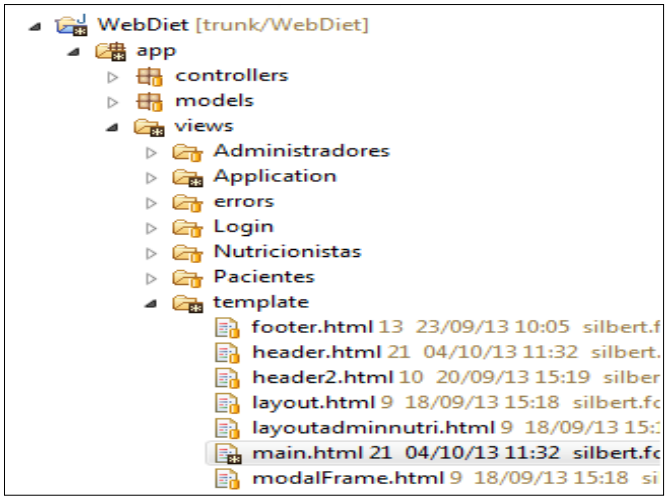
a) *Template footer.html*

b) *Template header.html*

c) *Template main.html*

A *template footer.html* é o cabeçalho das páginas e a *template header.html* o topo. Ambas são incluídas na *template main.html* – o modelo principal da aplicação. As outras *templates* presentes na aplicação são protótipos que foram criadas para teste (Figura 40).

Figura 40 - Estrutura de arquivos do WebDiet

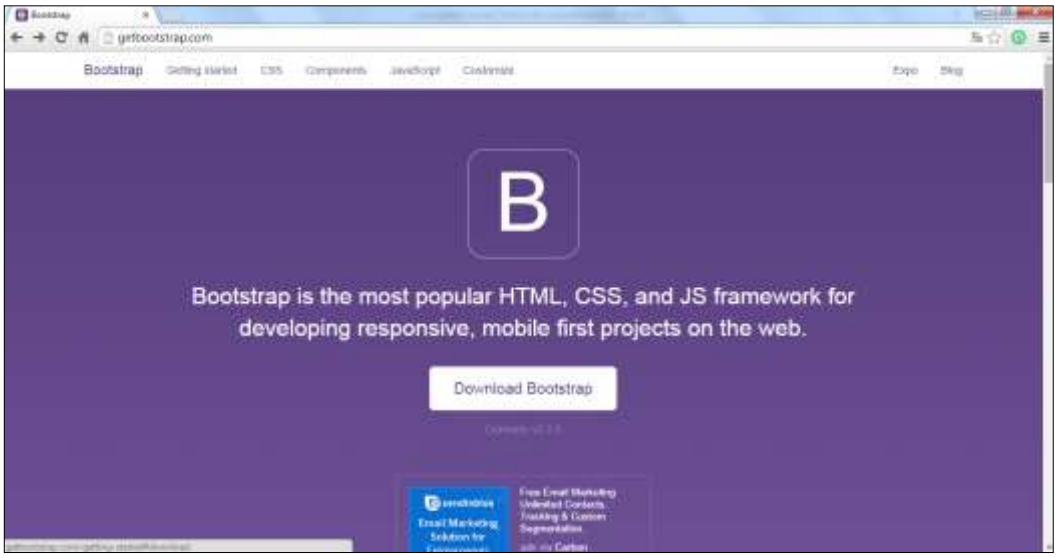



Fonte: Elaboração própria (2014).

### 3.4.1 Usando o *Bootstrap*

Para começar o desenvolvimento é necessário primeiro fazer o *download* dos arquivos do *framework Bootstrap* em seu site *getbootstrap.com* (Figura 41).

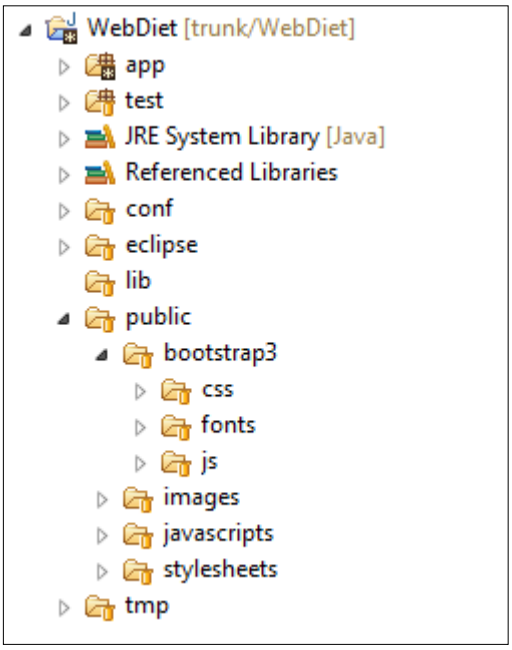
Figura 41 - Site do *framework Bootstrap*




Fonte: *Getbootstrap* (2014).

Depois de baixar o *Bootstrap*, o arquivo precisa ser descompactado e inserido no projeto. A estrutura de arquivos do *Bootstrap* são arquivos *Javascript*, folhas de estilos (CSS) e imagens prontas para ser utilizadas. No *Play* esses arquivos são importados para pasta *public* (Figura 42).

Figura 42 - *Bootstrap* na estrutura de diretórios do *projeto*




Fonte: Elaboração própria (2014).

## 3.5 *TEMPLATES* DO PROJETO WEBDIET

Esta seção mostra as *templates* que foram utilizadas no desenvolvimento do protótipo.

### 3.5.1 *Template header*

A *template header.html* é responsável pela formatação do topo das telas de exibição da aplicação. Página criada em HTML utilizando classes do *Bootstrap* para formatar a exibição da tela e funções do sistema de *templates* do Play para criação da lógica da aplicação. As Figuras 43 e 44 mostram o código fonte da *template header.html.*

Figura 43 - Código fonte da *template header.html* (Parte 1)

```
<!-- Fixed navbar -->
    <div class="navbar navbar-default navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
          <a class="navbar-brand" href="@{Application.index()}">WebDiet</a>
        </div>
        <div class="navbar-collapse collapse">
          <ul class="nav navbar-nav">
            <li class="active"><a href="@{Application.index()}"><span class="glyphicon glyphicon-home"></span> Home</a></li>
            #{if !session.pessoa}
            <li><a href="">About</a></li>
            <li><a href="">Contact</a></li>
            <li class="dropdown">
              <a href="" class="dropdown-toggle" data-toggle="dropdown">Dropdown <b class="caret"></b></a>
              <ul class="dropdown-menu">
                <li><a href="">Action</a></li>
                <li><a href="">Another action</a></li>
                <li><a href="">Something else here</a></li>
                <li class="divider"></li>
                <li class="dropdown-header">Nav header</li>
                <li><a href="">Separated link</a></li>
                <li><a href="">One more separated link</a></li>
              </ul>
            </li>
```

Fonte: Elaboração própria (2014).

Figura 44 - Código fonte da *template header.html* (Parte 2)

```
        #{/if}
      </ul>
      <ul class="nav navbar-nav navbar-right">
      #{if session.pessoa}
        <li><label></label></li>
        <li><a href="@{Login.logout()}" onclick="return confirm('Tem certeza que deseja sair da aplicação?')"><button class="btn btn-danger btn-xs">
        <span class="glyphicon glyphicon-off"></span>  Logout</button></a></li>
        #{/if}
        #{else}
        <li><a href="@{Login.index()}"><button class="btn btn-success btn-xs"><span class="glyphicon glyphicon-plus-sign"></span>  Login</button></a></li>
        #{/else}
      </ul>
    </div><!--/.nav-collapse -->
  </div>
</div>
```

Fonte: Elaboração própria (2014).

### 3.5.2 *Template footer*

A *template footer.html* é responsável pela formatação do cabeçalho das telas de exibição da aplicação. Página também criada em HTML (Figura 45).

Figura 45 - Código fonte da *template footer.html*

```
<hr>

    <footer>
      <p>© WebDiet</p>
    </footer>
```

Fonte: Elaboração própria (2014).

### 3.5.3 *Template main*

A *template* principal da aplicação, a *main.html* é responsável por ser o modelo da aplicação Webdiet onde praticamente todas as outras páginas de exibição herdam sua formatação e todas as outras características (Figura 46).

Figura 46 - Código fonte da *template* principal da aplicação *main.html*

```
<!DOCTYPE html>
<html>
    <head>
        <title>WebDiet</title>
            <meta charset="${_response_encoding}">
            <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/bootstrap.css'}">
            <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/bootstrap.min.css'}">
            <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/bootstrap-theme.css'}">
            <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/bootstrap-theme.min.css'}">
            <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/navbar-fixed-top.css'}">
        #{get 'moreStyles' /}
            <script src="@{'/public/javascripts/jquery-1.9.1.js'}" type="text/javascript" charset="UTF-8"></script>
            <script src="@{'/public/bootstrap3/js/bootstrap.js'}" type="text/javascript" charset="UTF-8"></script>
            <script src="@{'/public/bootstrap3/js/bootstrap.min.js'}" type="text/javascript" charset="UTF-8"></script>
            <script src="@{'/public/bootstrap3/js/modal.js'}" type="text/javascript" charset="UTF-8"></script>
            <script src="@{'/public/bootstrap3/js/dropdown.js'}" type="text/javascript" charset="UTF-8"></script>
        #{get 'moreScripts' /}

    </head>
    <body>
        <div class="container">
            #{include '/template/header.html' /}
            #{doLayout /}
            #{include '/template/footer.html' /}
        </div>
    </body>
</html>
```

Fonte: Elaboração própria (2014).

A estrutura da *template main.html* apresenta a importação dos arquivos necessários para a utilização do *Bootstrap* no elemento *link* e no *script.* No corpo da

página foram incluídas as outras duas *templates* da aplicação: a *header.html* e a *footer.html* – e também incluída a *tag doLayout* para indicar onde será exibido o conteúdo das páginas que herdam da *template*.

## 3.6 ESTUDO DE CASO

A *template* principal da aplicação é a *main.html*, onde a maioria das páginas de exibição estendem dela, inclusive a página inicial da aplicação mostrada na Figura 47. A *template main.html* também é denominada de página decoradora, onde uma outra página, como por exemplo a página *index.html* da aplicação WebDiet – recebe seus atributos através da utilização da *tag #{extends 'páginadecoradora.html'/}*. A Figura 47 contém o código fonte da página decorada *index.html*, que é a tela inicial da aplicação WebDiet.

Figura 47 - Código fonte da tela inicial da aplicação

```
#{extends 'template/main.html' /}
#{set title:'Home' /}



      <!-- Main component for a primary marketing message or call to action -->
      <div class="jumbotron">..

     <div class="row">..

<!-- START THE FEATURETTES -->

      <hr class="featurette-divider">

      <div class="row featurette">..

      <hr class="featurette-divider">

      <div class="row featurette">..

      <hr class="featurette-divider">

      <div class="row featurette">..
```

Fonte: Elaboração própria (2014).

A Figura 47 exibe a estrutura da página *index.html* que é decorada pela *template main.html*, com isso essa *template* assume a função de página decoradora principal da aplicação. Isso se torna evidente com o uso, presente no topo da figura - da *tag #{extends 'template/main.html'},* onde a tela inicial herda características como: formatação, informações e outras características da *template* principal.

Também foram utilizadas características do *Bootstrap* para formatação da página para exibição na tela. Como por exemplo, a classe *jumbotron* que foi utilizada no elemento *div* – que é responsável por já definir uma espécie de *banner* na página, ela se encontra no topo da página inicial da aplicação na cor cinza (Figura 39).

A *template* principal *main.html* também apresenta características bem especificas para herdar seus atributos. A Figura 48 apresenta o código fonte da *template* principal.

Figura 48 - Código fonte da *template* principal (*main.html*)

```
<!DOCTYPE html>

<html>
    <head>..
    <body>

    <div class="container">
    #{include '/template/header.html' /}

    #{doLayout /}

    #{include '/template/footer.html' /}
    </div>

    </body>
</html>
```

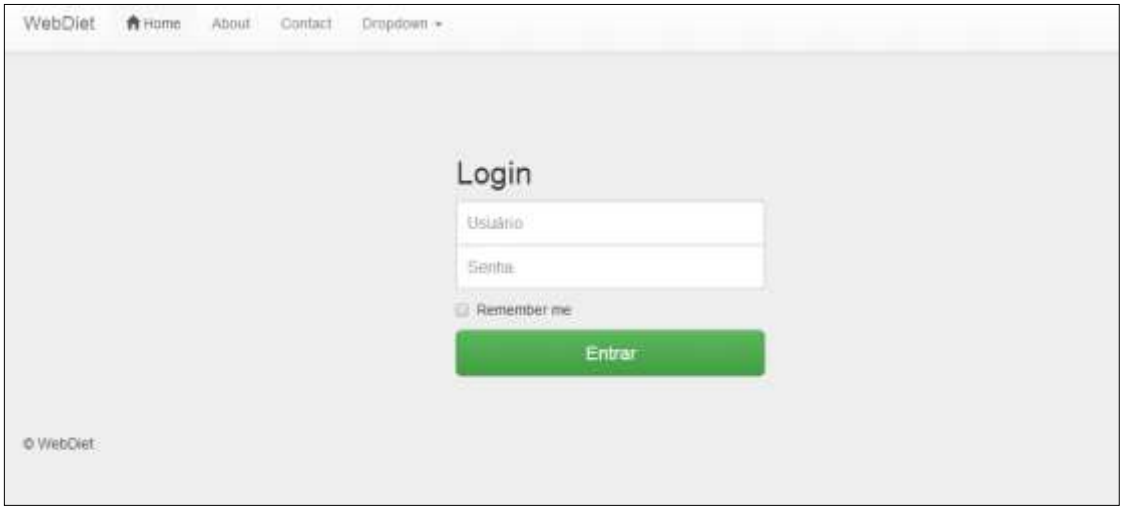
Fonte: Elaboração própria (2014).

A Figura 48 ilustra a estrutura da *template* principal *main.html*, onde foi preparada para ser uma página decoradora da aplicação. Isso se justifica com a utilização da *tag #{doLayout}*, onde especifica onde ficará as informações da página que vai herdar seus atributos. Em uma aplicação desenvolvida no *Play* poderá apresentar inúmeras páginas decoradoras, mas por padrão a *main.html* é utilizada como a principal.

Também é possível identificar que a *template* principal utiliza de duas outras *templates* da aplicação. Mas isso não deixa evidente uma herança, e sim uma simples inclusão. A página *main.html* está incluindo em seu corpo duas outras páginas para agregar informação através das *tags #{include '/template/header.html'/}* e *#{include '/template/footer.html'/}*. A *template header.html* foi construída para especificar o topo da tela inicial da aplicação e a *footer.html* o rodapé da tela inicial, contribuindo para a organização do projeto e assim evitando o acumulo excessivo de código em uma única página.

Para o usuário ter acesso a aplicação – tanto o nutricionista quando o administrador – ele precisa passar por uma tela de *login* para ter o devido acesso. A Figura 49 apresenta a tela responsável para entrar na aplicação WebDiet.

Figura 49 - Tela de *login*




Fonte: Elaboração própria (2014).

A figura 49 mostra a tela de *login* da aplicação, onde o nutricionista ou o administrador terão que passar para ter o devido acesso a aplicação. Foi utilizada a classe *"form-signin"* do *Bootstrap* para criar e formar a exibição desta tela através de um CSS especifico para a tela de acesso ao sistema, o arquivo *signin.css*. A Figura 50 exibe o código fonte da tela de *login* da aplicação.

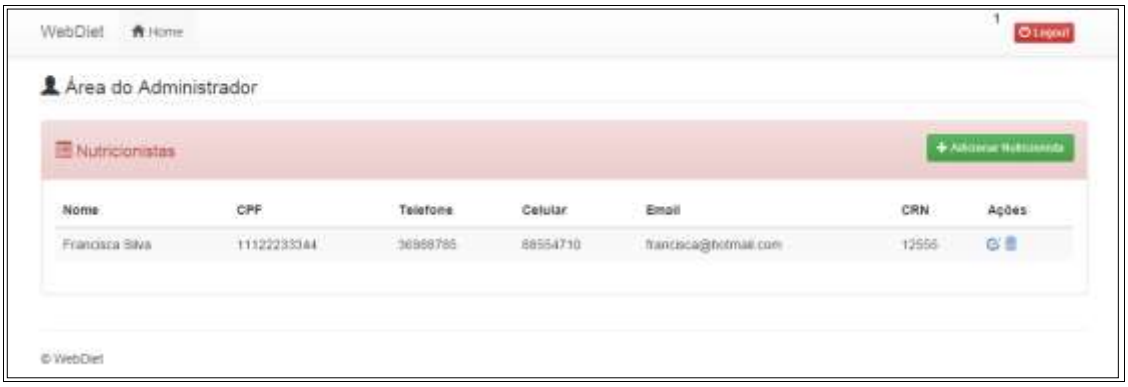
Figura 50 - Código fonte da tela de *login*

```
#{extends 'template/main.html' /}
#{set title:'Login' /}

 <link rel="stylesheet" media="screen" href="@{'/public/bootstrap3/css/signin.css'}">
<br/><br/><br/><br/>

      <form class="form-signin" action="@{Login.tryLogin()}">
        <h2 class="form-signin-heading">Login</h2>
        <input type="text" class="form-control" name="p.login"  placeholder="Usuário" autofocus="">
        <input type="password" class="form-control" name="p.senha"  placeholder="Senha">
        <label class="checkbox">
          <input type="checkbox" value="remember-me"> Remember me
        </label>
        <button class="btn btn-lg btn-success btn-block" type="submit">Entrar</button>
      #{ifErrors}<br/>
<div class="alert alert-danger">Senha ou Usuário Invalido!!!</div>

#{/ifErrors}
      </form>
```

Fonte: Elaboração própria (2014).

Depois de ser feita a identificação do usuário pelo sistema, ele será remetido para a área do administrador ou para a área do usuário. No caso do usuário administrador, o ambiente é caracterizado por gerenciar as informações do nutricionista. A tela de exibição da área do administrador e suas principais funcionalidades estão ilustradas na Figura 51.

Figura 51 - Área do administrador




Fonte: Elaboração própria (2014).

A Figura 51 exibe a tela de uso exclusivo do administrador da aplicação, no qual pode adicionar o nutricionista, editar e excluir - além de visualizar os nutricionistas cadastrados.

Nessa tela foi utilizado uma funcionalidade importante do sistema de *template* do *Play*, que são as expressões, apresentadas na Figura 52.

Figura 52 - Código fonte da área do administrador

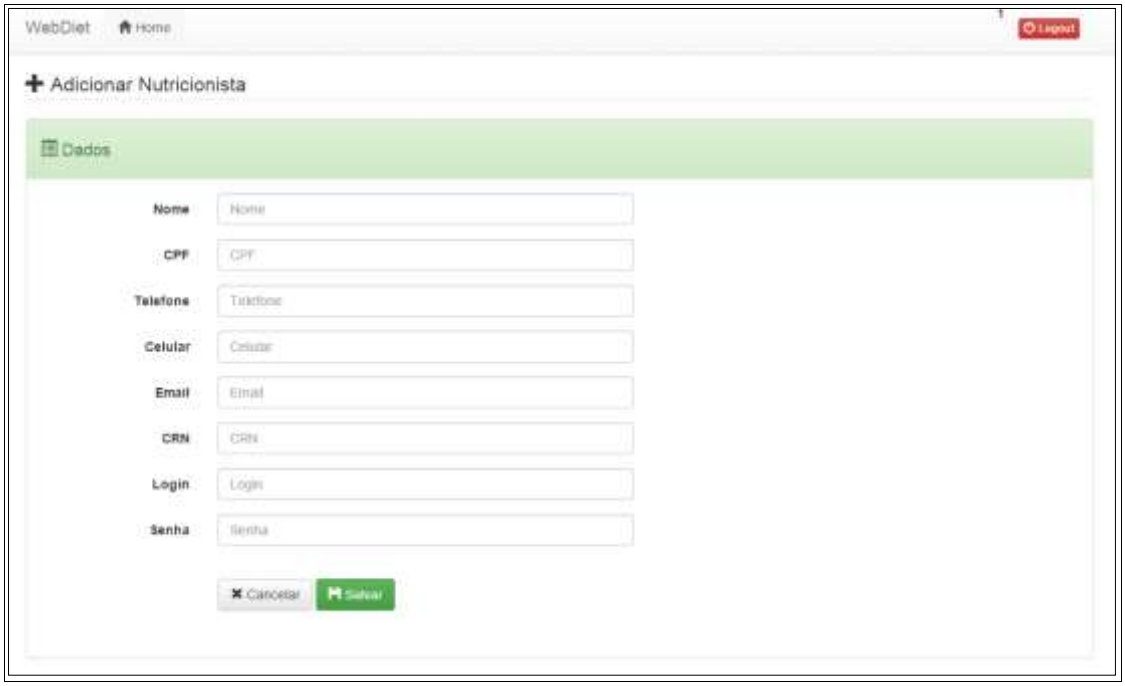
```
    #{if !nutricionistas.empty}
<table class="table table-striped">
                <thead>
                    <tr>
                        <th>Nome</th>
                        <th>CPF</th>
                        <th>Telefone</th>
                        <th>Celular</th>
                        <th>Email</th>
                        <th>CRN</th>
                        <th>Ações</th>
                    </tr>
                </thead>
                <tbody>
                    #{list items:nutricionistas, as: 'nutricionista'}
                    <tr>
                        <td>${nutricionista.nome}</td>
                        <td>${nutricionista.cpf}</td>
                        <td>${nutricionista.telefone}</td>
                        <td>${nutricionista.celular}</td>
                        <td>${nutricionista.email}</td>
                        <td>${nutricionista.crn}</td>
                        <td><a data-toggle="modal" href="@{nutricionistas.show(nutricionista.id)}"
                            <a href="@{nutricionistas.excluir(nutricionista.id)}"
                            </td>
                    </tr>
```

Fonte: Elaboração própria (2014).

A Figura 52 consta o código fonte da tela inicial da área do administrador, onde são utilizadas as expressões para exibir informações do (a) nutricionista. Para exibir as informações foram utilizadas as seguintes expressões: *${nutricionista.nome}, ${nutricionista.cpf}, ${nutricionista.telefone}, ${nutricionista.celular}, ${nutricionista.email}* e *${nutricionista.crn}* – onde respectivamente exibe o nome, Cadastro de Pessoas Físicas (CPF), telefone, celular, *email* e o código Conselho Regional de Nutrição (CRN) do nutricionista.

Na área do administrador também é possível adicionar um novo nutricionista na aplicação, conforme ilustrado na Figura 53.

Figura 53 - Adicionar Nutricionista




Fonte: Elaboração própria (2014).

A Figura 53 mostra o formulário de informações que são necessárias para adicionar um nutricionista na aplicação. As informações como: nome, CPF, telefone, celular, *email*, CRN, *login* e senha são todos campos obrigatórios.

Figura 54 - Código fonte do formulário adicionar nutricionista parte 1

```
#{extends 'template/main.html' /}
#{set title:'Nutricionista' /}


<fieldset>
<legend><span class="glyphicon glyphicon-plus"></span> #{if nutricionista?.id}Editar #{/if}
    #{else}Adicionar#{/else} Nutricionista</legend>

<div class="panel panel-success">
  <div class="panel-heading">
    <h4><span class="glyphicon glyphicon-list-alt"></span> Dados</h4>
  </div>
  <div class="panel-body">
<form class="form-horizontal" role="form" action="@{Nutricionistas.salvar()}">
 <input type="hidden" name="nutricionista.id" value="${nutricionista?.id}" />
  <div class="form-group">
    <label for="inputNome1" class="col-lg-2 control-label">Nome</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputNome1" placeholder="Nome" name="nutricionista.nome" value="${nutricionista?.nome}">
    </div>
    <span class="label label-danger">#{error 'nutricionista.nome' /}</span>
  </div>
  <div class="form-group">
    <label for="inputCpf1" class="col-lg-2 control-label">CPF</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputCpf1" placeholder="CPF" name="nutricionista.cpf" value="${nutricionista?.cpf}">
   </div>
   <span class="label label-danger">#{error 'nutricionista.cpf' /}</span>
  </div>
  <div class="form-group">
    <label for="inputTelefone1" class="col-lg-2 control-label">Telefone</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputTelefone1" placeholder="Telefone" name="nutricionista.telefone" value="${nutricionista?.telefone}">
   </div>
   <span class="label label-danger">#{error 'nutricionista.telefone' /}</span>
  </div>
```

Fonte: Elaboração própria (2014).

As Figuras 54 e 55 exibem o código fonte do formulário para adicionar nutricionista. Para a implementação foram utilizados dos artifícios do sistema de *template* do *Play* e de elementos *Bootstrap* para formatação. Além do uso de uma *action @{Nutricionistas.salvar()}* - responsável por criar o novo objeto e guardar no banco de dados da aplicação - e de expressões (*${nutricionista?.nome}*) também foi utilizado um elemento *error* que é utilizado para validação de erros na aplicação.

Figura 55 - Código fonte do formulário adicionar nutricionista parte 2

```
<div class="form-group">
  <label for="inputCelular1" class="col-lg-2 control-label">Celular</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputCelular1" placeholder="Celular" name="nutricionista.celular" value="${nutricionista?.celular}">
  </div>
  <span class="label label-danger">#{error 'nutricionista.celular' /}</span>
</div>
<div class="form-group">
  <label for="inputEmail1" class="col-lg-2 control-label">Email</label>
  <div class="col-lg-5">
    <input type="email" class="form-control" id="inputEmail1" placeholder="Email" name="nutricionista.email" value="${nutricionista?.email}">
  </div>
  <span class="label label-danger">#{error 'nutricionista.email' /}</span>
</div>
<div class="form-group">
  <label for="inputCrn1" class="col-lg-2 control-label">CRN</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputCrn1" placeholder="CRN" name="nutricionista.crn" value="${nutricionista?.crn}">
  </div>
  <span class="label label-danger">#{error 'nutricionista.crn' /}</span>
</div>
<div class="form-group">
  <label for="inputLogin1" class="col-lg-2 control-label">Login</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputLogin1" placeholder="Login" name="nutricionista.login" value="${nutricionista?.login}">
  </div>
  <span class="label label-danger">#{error 'nutricionista.login' /}</span>
</div>
<div class="form-group">
  <label for="inputSenha1" class="col-lg-2 control-label">Senha</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputSenha1" placeholder="Senha" name="nutricionista.senha" value="${nutricionista?.senha}">
  </div>
  <span class="label label-danger">#{error 'nutricionista.senha' /}</span>
</div><br/>
<div class="form-group">
  <div class="col-lg-offset-2 col-lg-10">
    <a href="@{Administradores.index()}" class="btn btn-default"><span class="glyphicon glyphicon-remove"></span> Cancelar</a>
    <button type="submit" class="btn btn-success"><span class="glyphicon glyphicon-floppy-disk"></span> Salvar</button>
  </div>
</div>
<br/>
</form>
</div>
</div>
</fieldset>
```

Fonte: Elaboração própria (2014).

O ambiente do nutricionista, além de gerenciar informações do paciente, como adicionar, editar, excluir e visualizar informações do paciente, também apresenta funcionalidades para o auxílio do nutricionista como: calcular IMC e criação de dietas (Figura 56).

Figura 56 - Área do nutricionista


Fonte: Elaboração própria (2014).

Figura 57 - Código fonte da área do nutricionista

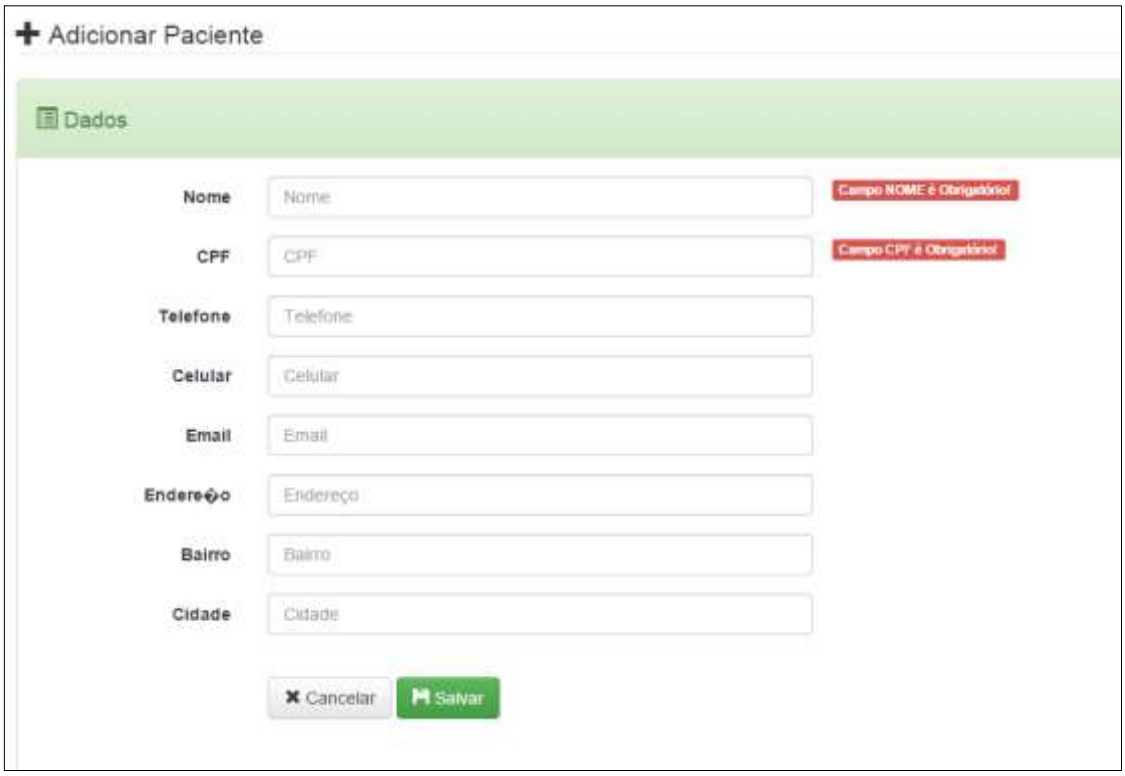
```
#{extends 'template/main.html' /}
#{set title:'areaNutri' /}

<fieldset>
<legend><span class="glyphicon glyphicon-user"></span> Área do Nutricionista</legend>

<div id="botoesMenu">

<div class="panel panel-success">
  <div class="panel-heading">
    <h4><span class="glyphicon glyphicon-list-alt"></span> Pacientes</h4>
  </div>
  <div class="panel-body">
    #{if !pacientes.empty}
<table class="table table-striped">
                <thead>
                    <tr>
                        <th>Nome</th>
                        <th>CPF</th>
                        <th>Telefone</th>
                        <th>Celular</th>
                        <th>Email</th>
                        <th>Ações</th>
                    </tr>
                </thead>
                <tbody>
                    #{list items:pacientes, as: 'paciente'}
                    <tr>
                        <td>${paciente.nome}</td>
                        <td>${paciente.cpf}</td>
                        <td>${paciente.telefone}</td>
                        <td>${paciente.celular}</td>
                        <td>${paciente.email}</td>
                        <td><a data-toggle="modal" href="@{pacientes.show(paciente.id)}"
                            class="glyphicon glyphicon-edit" title="Editar"></a>
                            <a href="@{pacientes.excluir(paciente.id)}"
                            class="glyphicon glyphicon-trash" title="Excluir" onclick="return confirm('Tem certeza que deseja excluir?')"></a>
                            </td>
                    </tr>
                    #{/list}
                </tbody>
            </table>
```

Fonte: Elaboração própria (2014).

A figura 57 contém o código fonte da tela inicial da área do nutricionista. E para exibir as informações do paciente foram utilizadas as expressões: *${paciente.nome}, ${paciente.cpf}, ${paciente.telefone}, ${paciente.celular}* e *${paciente.email}* – onde respectivamente exibe o nome, CPF, telefone, celular e o *email* do paciente. Foram empregadas também *actions*, responsáveis por editar e excluir um paciente, respectivamente *@{pacientes.show(paciente.id)}* e *@{pacientes.excluir(paciente.id)}*.

Na área do nutricionista também é possível adicionar um novo paciente. A Figura 58 mostra a tela de exibição responsável por inserir um paciente no sistema.

Figura 58 - Adicionar Paciente




Fonte: Elaboração própria (2014).

A Figura 58 apresenta o formulário para inserção de um paciente no sistema. As informações necessárias para o cadastro são: nome, CPF, telefone, celular, *email*, endereço, bairro e cidade – onde apenas nome e CPF são obrigatórios.

Figura 59 - Código fonte do formulário adicionar paciente parte 1

```
#{extends 'template/main.html' /}
#{set title:'Paciente' /}


<fieldset>
<legend><span class="glyphicon glyphicon-plus"></span> #{if paciente?.id}Editar #{/if}
    #{else}Adicionar#{/else} Paciente</legend>

<div class="panel panel-success">
  <div class="panel-heading">
    <h4><span class="glyphicon glyphicon-list-alt"></span> Dados</h4>
  </div>
  <div class="panel-body">
<form class="form-horizontal" role="form" action="@{Pacientes.salvar()}">
 <input type="hidden" name="paciente.id" value="${paciente?.id}" />
  <div class="form-group">
    <label for="inputNome1" class="col-lg-2 control-label">Nome</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputNome1" placeholder="Nome" name="paciente.nome" value="${paciente?.nome}">
    </div>
    <span class="label label-danger">#{error 'paciente.nome' /}</span>
  </div>
  <div class="form-group">
    <label for="inputCpf1" class="col-lg-2 control-label">CPF</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputCpf1" placeholder="CPF" name="paciente.cpf" value="${paciente?.cpf}">
    </div>
    <span class="label label-danger">#{error 'paciente.cpf' /}</span>
  </div>
  <div class="form-group">
    <label for="inputTelefone1" class="col-lg-2 control-label">Telefone</label>
    <div class="col-lg-5">
      <input type="text" class="form-control" id="inputTelefone1" placeholder="Telefone" name="paciente.telefone" value="${paciente?.telefone}">
    </div>
    <span class="label label-danger">#{error 'paciente.telefone' /}</span>
  </div>
```

Fonte: Elaboração própria (2014).

Figura 60 - Código fonte do formulário adicionar paciente parte 2

```
<div class="form-group">
  <label for="inputCelular1" class="col-lg-2 control-label">Celular</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputCelular1" placeholder="Celular" name="paciente.celular" value="${paciente?.celular}">
  </div>
  <span class="label label-danger">#{error 'paciente.celular' /}</span>
</div>
<div class="form-group">
  <label for="inputEmail1" class="col-lg-2 control-label">Email</label>
  <div class="col-lg-5">
    <input type="email" class="form-control" id="inputEmail1" placeholder="Email" name="paciente.email" value="${paciente?.email}">
  </div>
  <span class="label label-danger">#{error 'paciente.email' /}</span>
</div>
<div class="form-group">
  <label for="inputEndereco1" class="col-lg-2 control-label">Endereco</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputEndereco1" placeholder="EndereÃ§o" name="paciente.endereco" value="${paciente?.endereco}">
  </div>
  <span class="label label-danger">#{error 'paciente.endereco' /}</span>
</div>
<div class="form-group">
  <label for="inputBairro1" class="col-lg-2 control-label">Bairro</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputBairro1" placeholder="Bairro" name="paciente.bairro" value="${paciente?.bairro}">
  </div>
  <span class="label label-danger">#{error 'paciente.bairro' /}</span>
</div>
<div class="form-group">
  <label for="inputCidade1" class="col-lg-2 control-label">Cidade</label>
  <div class="col-lg-5">
    <input type="text" class="form-control" id="inputCidade1" placeholder="Cidade" name="paciente.cidade" value="${paciente?.cidade}">
  </div>
  <span class="label label-danger">#{error 'paciente.cidade' /}</span>
</div>
<br/>
<div class="form-group">
  <div class="col-lg-offset-2 col-lg-10">
    <a href="@{Nutricionistas.index()}" class="btn btn-default"><span class="glyphicon glyphicon-remove"></span> Cancelar</a>
    <button type="submit" class="btn btn-success"><span class="glyphicon glyphicon-floppy-disk"></span> Salvar</button>
  </div>
</div>
<br/>
</form>
```

Fonte: Elaboração própria (2014).

As Figuras 59 e 60 representam o código fonte da página de formulário para adicionar um paciente. Essa página apresenta: a *action @{Pacientes.salvar()}* responsável para adição do objeto paciente na aplicação; expressões como *${paciente?.nome}* para adicionar ou editar informações do paciente; e do elemento *error* para validar essas informações.

# 4 CONSIDERAÇÕES FINAIS

O projeto desenvolvido neste trabalho permitiu mostrar a utilização do sistema de *template*s do *Play* bem como do *Bootstrap* para desenvolvimento mais rápido e produtivo das páginas *web* de um projeto. A principal vantagem remete ao reuso do código, que se torna possível através das características de herança de *templates* e da sintaxe do *Play*.

## 4.1 TRABALHOS FUTUROS

Como trabalho futuro, sugere-se o desenvolvimento total da aplicação WebDiet para plataforma *web*, com o fim de contribuir no trabalho do profissional da área de nutrição e da saúde do paciente.

Outra proposta é o desenvolvimento da plataforma *mobile*, para somente o paciente consultar suas dietas criadas por seu nutricionista.

# REFERÊNCIAS

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

**GETTING started**. Disponível em:<http://getbootstrap.com/getting-started>. Acesso em: 05 out. 2014.

LEROUX, Nicolas; KAPER, Sietse de. **Play for Java:** Covers Play 2. New York: Manning, 2012.

ZENEXTY; TYPESAFE; **Play 1.2.x Documentation**. Disponível em:<https://www.playframework.com/documentation/1.2.x/home>. Acesso em: 05 out. 2014.