

**UNIVERSIDADE FUMEC
FACULDADE DE CIÊNCIAS EMPRESARIAIS
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**JOÃO PAULO CARNEIRO ARAMUNI
RAFAEL ANDRADE LOTT
ROBERTO JOSÉ**

PLAY FRAMEWORK

**BELO HORIZONTE
2013**

UNIVERSIDADE FUMEC
FACULDADE DE CIÊNCIAS EMPRESARIAIS
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PAULO CARNEIRO ARAMUNI
RAFAEL ANDRADE LOTT
ROBERTO JOSÉ

PLAY FRAMEWORK

Atividade Auto Instrucional apresentado ao Professo Flávio Velloso Laper como requisito parcial para aprovação na disciplina Desenvolvimento em Ambiente Web do curso de Ciência da Computação da Universidade FUMEC.

BELO HORIZONTE
2013

SUMÁRIO

1.	INTRODUÇÃO	4
2.	MOTIVAÇÃO	5
3.	DIFERENCIAIS	6
4.	COMPONENTES.....	7
5.	FUNCIONALIDADES EVIDENTES.....	8
5.1.	MANIPULAÇÃO DE ERROS.....	8
5.2.	PADRONIZAÇÃO MVC.....	8
5.3.	ROTEAMENTO REST.....	9
5.4.	PERSISTÊNCIA DE DADOS (JPA).....	10
5.5.	LOCALE	10
6.	APLICAÇÃO	11
6.1.	PLAY CONSOLE.....	11
6.2.	CRIAÇÃO DE APLICAÇÃO.....	11
6.3.	DATABASE	20
6.4.	GERAR ARQUIVO JAR	22
6.5.	PUBLICAR	23
7.	CONCLUSÃO.....	26
8.	LINKS ÚTEIS	27
8.1.	LINKS GERAIS	27
8.2.	LINKS HEROKU: CLOUD APPLICATION PLATFORM	27
8.3.	LINK DA APLICAÇÃO DESENVOLVIDA	28
9.	REFERÊNCIAS.....	28

1. INTRODUÇÃO

Para atingir alta produtividade no desenvolvimento Web, está surgindo uma série de frameworks construídos a partir de conceitos como:

- Convention Over Configuration;
- Kiss (Keep It Simple, Stupid!).

Isso significa que, se o desenvolvedor seguir as convenções ditadas pelo framework, o mesmo será capaz de inferir as ações a serem tomadas, sem a necessidade de configurações adicionais, como XML, Annotations ou qualquer outro tipo de meta-dado.

Além disso, o framework pode gerar vários templates pré-definidos de aplicações e boas práticas, como suporte a testes unitários, etc.

Ao poupar o programador de configurações complicadas, do gerenciamento de arquivos XML, e de boa parte do que se refere à infraestrutura, o mesmo pode se focar mais no core bussiness da aplicação, com efetivo ganho de qualidade e produtividade.

O **Play** é um framework open source para aplicações Web, escrito em Java, que possibilita o desenvolvimento de aplicações Web que seguem o padrão MVC. Tem por objetivo otimizar a produtividade do desenvolvedor através do uso de configuração sobre convenção (CoC). Com recompilação feita durante a execução da aplicação, e caso ocorra algum erro, o respectivo é exibido no browser, indicando a linha do erro.

Apesar de o Play ter sido escrito em Java, ele suporta a linguagem Scala desde a versão 1.1 (Atual: 2.1.1). A empresa Typesafe (responsável pela linguagem Scala), anunciou a aquisição do Play Framework e este será mantido pela empresa. Uma das novidades relacionadas a este anúncio é a reescrita do núcleo do Play Framework, na versão 2.0, totalmente em Scala.

2. MOTIVAÇÃO

O Play foi muito inspirado no Ruby on Rails e Django. Um desenvolvedor familiarizado com qualquer um desses frameworks, irá se adaptar ao Play com facilidade.

Ruby on Rails e Django são frameworks produtivos, então por que não ter o mesmo nível de produtividade com **Java**?

O Play Framework utiliza do poder das aplicações Java, porém sem a burocracia necessária para o desenvolvimento de aplicações centradas no modelo Java Enterprise. Libertando das metodologias e ideologias relacionadas ao desenvolvimento de aplicações Java EE, o Play provê para os desenvolvedores uma maneira fácil e elegante de trabalhar, visando o aumento da produtividade.

Basta um editor de texto e será mais que o suficiente para o desenvolvimento de aplicações, chega a ser incrível pensar que conseguiremos desenvolver aplicações Java Web sem a necessidade de um IDE (Eclipse, Netbeans,...), mas vale lembrar, que estas IDE's ainda possuem seus atrativos e auxiliam na produtividade do desenvolvedor.

Apesar de que as aplicações desenvolvidas com o Play, foram projetadas para executar dentro do JBoss Netty Web Server, as aplicações podem ser empacotadas em arquivos WAR e distribuídas para outros servidores de aplicações Java EE (ex.: Apache Tomcat).

3. DIFERENCIAIS

Diferenciais do Play em relação aos demais frameworks:

Stateless: O Play é totalmente RESTful – não existe conexão por sessão Java EE. Isto torna o Play muito mais escalável que os demais frameworks.

Sem configuração: realizar o download, descompactar e desenvolver.

Fácil ida e volta: sem necessidade de deploy no servidor de aplicação, apenas edite o código e atualize o browser.

Teste unitário integrado: suportes nativos para JUnit e Selenium.

API elegante: raramente um desenvolvedor terá a necessidade de importar alguma lib. O Play já disponibiliza a maioria dos recursos necessários para o desenvolvimento de uma aplicação.

Métodos estáticos: todos os controles de entrada e métodos de negócio são declarados como estáticos. E isto é de fato bem diferente do que vemos nos demais frameworks Java.

I/O Assíncrona: através do uso do servidor web JBoss Netty, o Play consegue disponibilizar e tratar uma enorme quantidade de requisições assíncronas.

Arquitetura Modular: assim como Rail e Django, o Play utiliza o conceito de módulos. O que possibilita um meio elegante e simples de expandir o core do Play.

Módulo CRUD: fácil construção de UI administrativas com pouco código.

Módulo Scala: disponibiliza um suporte completo para Scala.

4. COMPONENTES

O Play utiliza massivamente algumas bibliotecas populares:

- ✓ JBoss Netty para o servidor web.
- ✓ Hibernate para a camada de dados.
- ✓ Groovy para a os templates.
- ✓ O compilador do Eclipse para atualização da aplicação sem necessidade de realizar um deploy da aplicação para testar as alterações (hot-reloading).
- ✓ Apache Ivy para gerenciamento de dependências.

Funcionalidades presentes no núcleo do Play:

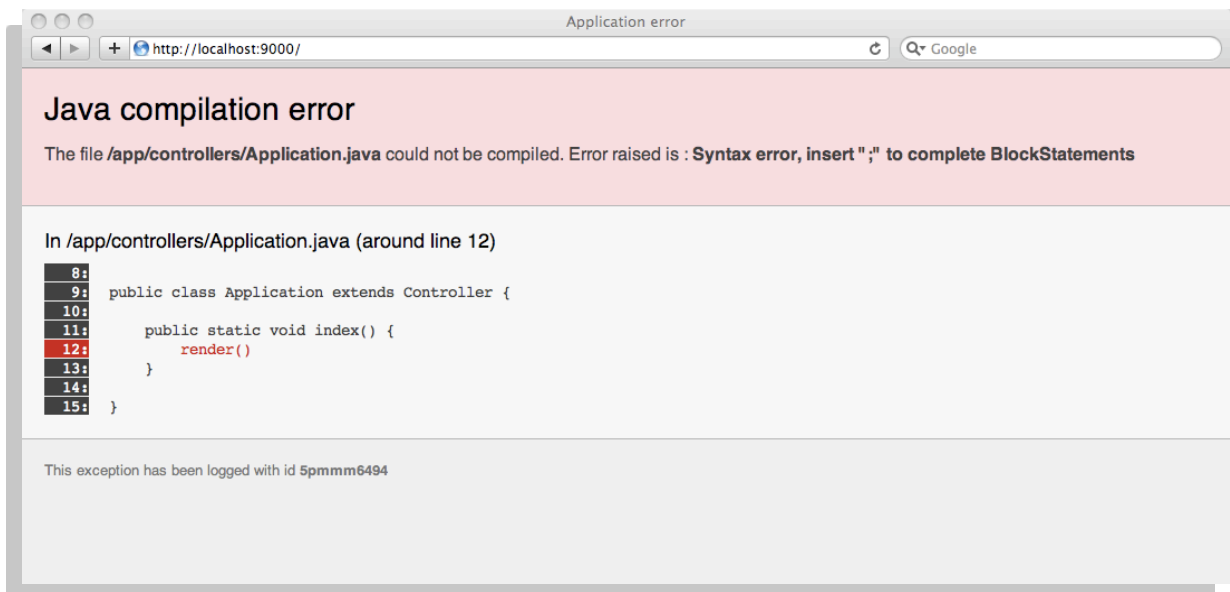
- Um framework RESTful limpo e leve.
- CRUD: um módulo para simplificar a edição de modelos de objetos.
- Secure: um módulo para habilitar um sistema de autenticação de usuários.
- Um framework de validação baseado em anotações.
- Um Job Scheduler (agendamento de tarefas).
- Suporte para emails SMTP de maneira simples.
- Suporte para JSON e XML.
- Uma camada de persistência baseada em JPA.
- Uma base de dados embutida para rápido deploy e testes da aplicação.
- Um framework completo para realização de testes.
- Funcionalidade para upload de arquivos.
- Suporte para múltiplos ambientes de desenvolvimento.
- Poderosa engine de templates baseadas em Groovy com hierarquia e tags.
- Arquitetura modular que possibilita criar novas funcionalidades para o núcleo.
- Suporte para OpenID e clientes Web Service.

5. FUNCIONALIDADES EVIDENTES

Antes de usar o Play Framework, vamos ver algumas de suas funcionalidades.

5.1. MANIPULAÇÃO DE ERROS

Quando existe algum erro de sintaxe no código, ao acessar o sistema, teremos uma tela semelhante a esta:



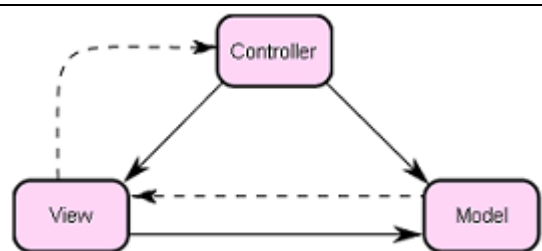
Com o erro na tela, podemos facilmente corrigir e recarregar a página.

Não é preciso recompilar nada!

5.2. PADRONIZAÇÃO MVC

Seguindo o padrão MVC, o seu projeto consiste de três pastas: *app/controllers*, *app/models* e *app/views*. Cada controller possui uma pasta em *app/views* e cada método do controller geralmente possui arquivo template na pasta.

Um diagrama simples exemplificando a relação entre Model, View e Controller. As linhas sólidas indicam associação direta e as tracejadas indicam associação indireta.



Por exemplo, o controller **Application** (criado pelo Play!) possui o método **index()**, que por sua vez possui o template *app/views/Application/index.html*.

Vamos ver este controller: *app/controllers/Application.java*

```
package controllers;
import play.*;
import play.mvc.*;
import java.util.*;
import models.*;
public class Application extends Controller {
    public static void index() {
        render();
    }
}
```

O **render()** irá carregar o template com o nome do método.

O template é exibido logo a seguir: *app/views/Application/index.html*

5.3. ROTEAMENTO REST

O arquivo */conf/routes* contém todo o roteamento RESTful da aplicação:

```
# Routes
# This file defines all application routes (Higher priority routes first)
# Home page
GET /Application.index
# USERS
GET /user/{id}
Users.show
# Ignore favicon requests
GET /favicon.ico
404
# Map static resources from the /app/public folder to the /public path
GET /public/staticDir:public
# Catch all
*/{controller}/{action}{controller}.{action}
```

O primeiro item do roteamento diz que a raiz do site irá carregar o método `index` do controller `Application`.

Outro exemplo é quando acessamos `/user/1`, redirecionando para o controller `Users` e o método `show`. Veja que temos o roteamento baseado no **GET**, mas podemos ter algo como **POST**, **PUT**, **DELETE** etc.

5.4. PERSISTÊNCIA DE DADOS (JPA)

Um dos fatores principais de qualquer framework é persistir dados no banco relacional de forma orientada a objetos. O Play!, juntamente com o JPA, resolve esse problema facilmente. Basta criar as classes usando a nomenclatura JPA e realizar as operações para manipulação de dados.

5.5. LOCALE

Suporte para a tradução de mensagens de forma fácil, bastando apenas usar as template tags disponíveis.

Ainda existem outras funcionalidades interessantes ligadas ao **Play!**

6. APLICAÇÃO

6.1. PLAY CONSOLE

Para quem está acostumado com os frameworks tradicionais do mercado, deve imaginar que é necessário criar um projeto Web em um IDE, importar alguns arquivos jars, configurar certos arquivos e escrever a aplicação, certo? Errado! Com o Play Framework esse processo se torna incrivelmente simples.

Para criar uma nova aplicação, usa-se um utilitário, chamado **Play Console**, que acompanha o framework:

Notas:

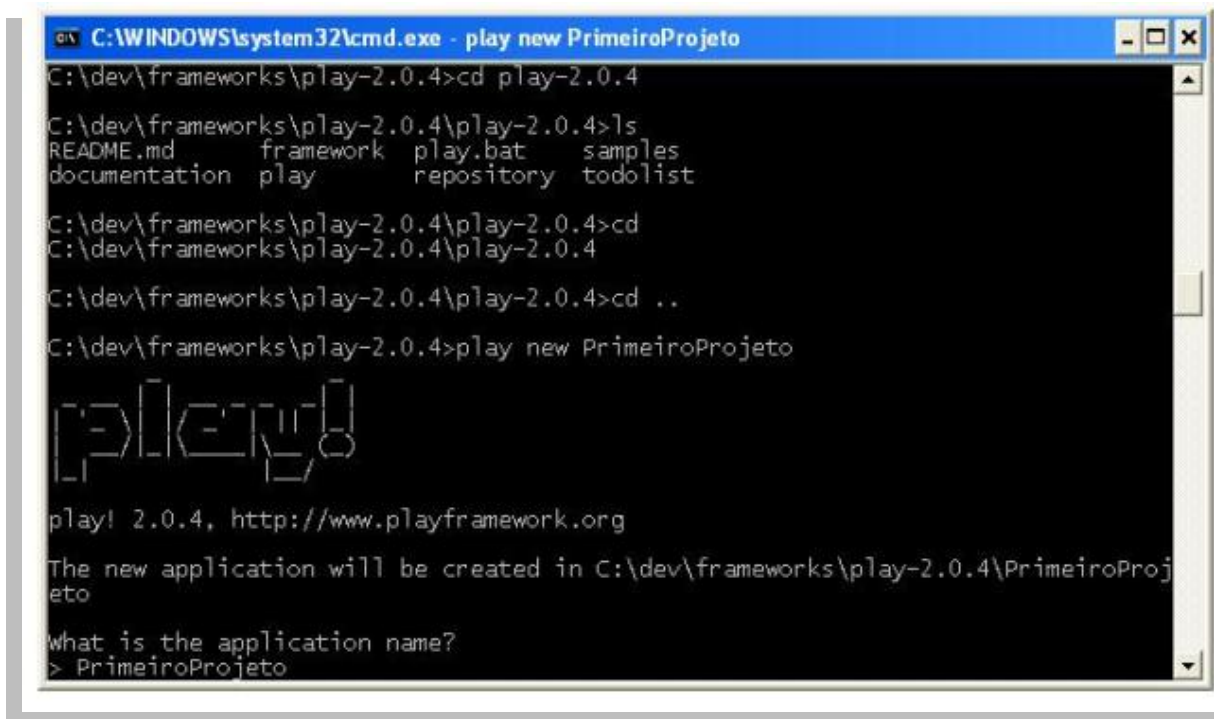
- ✓ Para executar o Play! Framework, é necessário o JDK 6 ou superior. Além disso, configurar o diretório bin do JDK na variável de ambiente PATH do Sistema Operacional (O Play! Framework irá utilizar o javac, além do java);
- ✓ Incluir o diretório que contém os programas play e play.bat também na variável PATH do Sistema Operacional. (Na, *Figura 1*, o caminho será *C:\dev\frameworks\play-2.0.4\play-2.0.4*);
- ✓ No caso do Linux, executar o comando `chmod` no arquivo `play`.
- ✓ Outro comando importante é o gerador de war (deploy), dado por:
 - `play war nomeDoProjeto -o nomeDoProjeto.war`

6.2. CRIAÇÃO DE APLICAÇÃO

Criando o primeiro projeto com Play!

Utilizando o cmd no Windows (ou shell no Linux):

-
- `play new PrimeiroProjeto`
-
- `play eclipsify PrimeiroProjeto` (Se deseja usar o Eclipse)
-
- `play netbeansify PrimeiroProjeto` (Se deseja usar o NetBeans)
-

Figura 1 – Definindo o **Nome** do Projeto

```
C:\WINDOWS\system32\cmd.exe - play new PrimeiroProjeto
C:\dev\frameworks\play-2.0.4>cd play-2.0.4
C:\dev\frameworks\play-2.0.4\play-2.0.4>ls
README.md    framework    play.bat     samples
documentation play         repository   todoclist

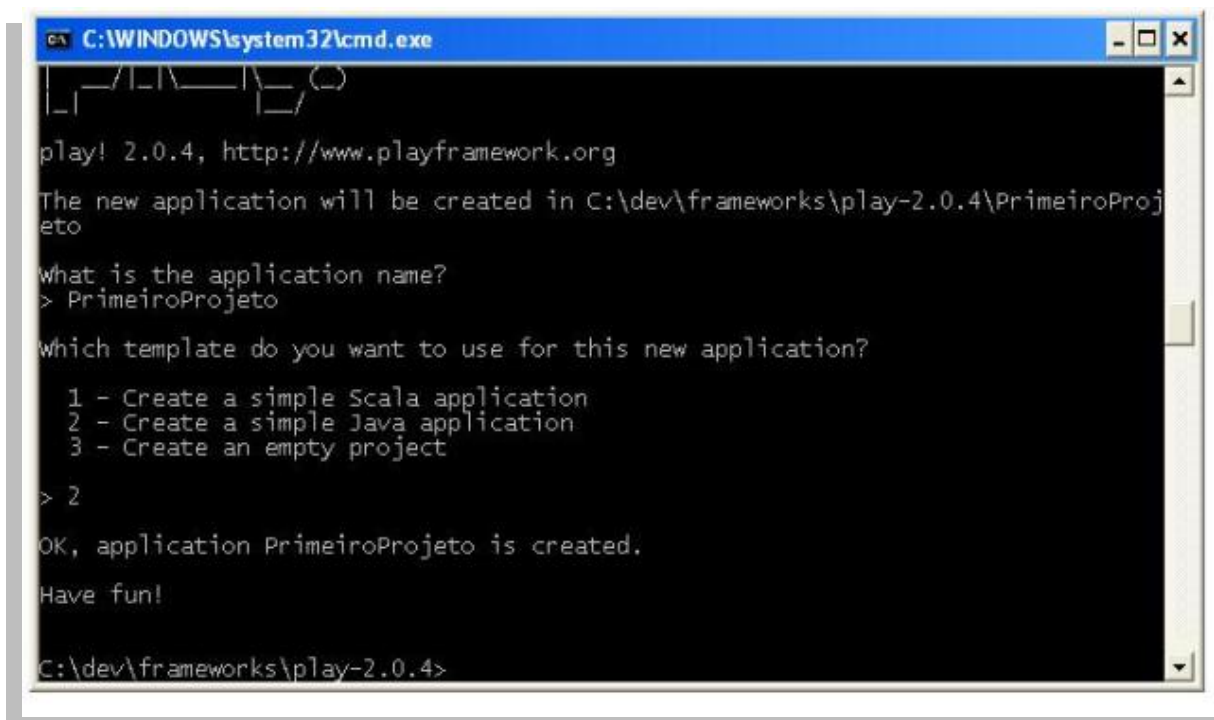
C:\dev\frameworks\play-2.0.4\play-2.0.4>cd
C:\dev\frameworks\play-2.0.4\play-2.0.4>cd ..
C:\dev\frameworks\play-2.0.4>play new PrimeiroProjeto

play! 2.0.4, http://www.playframework.org

The new application will be created in C:\dev\frameworks\play-2.0.4\PrimeiroProjeto

What is the application name?
> PrimeiroProjeto
```

No próximo passo, o console irá perguntar que template de projeto deverá ser criado, em Scala (opção 1), Java (opção 2) ou Empty Project (opção 3). Seguiremos com **Java** ! (opção 2).

Figura 2 – Definindo o **Tipo** de Projeto

```
C:\WINDOWS\system32\cmd.exe
play! 2.0.4, http://www.playframework.org

The new application will be created in C:\dev\frameworks\play-2.0.4\PrimeiroProjeto

What is the application name?
> PrimeiroProjeto

Which template do you want to use for this new application?

1 - Create a simple Scala application
2 - Create a simple Java application
3 - Create an empty project

> 2

OK, application PrimeiroProjeto is created.

Have fun!

C:\dev\frameworks\play-2.0.4>
```

Nesse momento, o esqueleto da aplicação foi criado.

Como o comando play foi executado do diretório `C:\dev\frameworks\play-2.0.4`, nesse mesmo local será criado o diretório `PrimeiroProjeto`, que contém a seguinte estrutura:

- **app** - contêm os models, controllers e views do projeto;
- **conf** - contêm os arquivos de configuração como o arquivo de rotas (routes), configurações gerais da aplicação (application.conf), internacionalização, etc.;
- **project** - contêm os build scripts do projeto (os scripts são criados para a ferramenta sbt, que é uma ferramenta de build para Scala e Java). Raramente você terá que alterá-lo;
- **public** - contêm recursos como arquivos javascript, css e imagens.

O próximo passo é rodar a aplicação. Para isso, basta entrar no diretório `PrimeiroProjeto` via linha de comando e digitar play novamente. O play console irá aguardar comandos. Então é só digitar **run** e pronto, a aplicação estará no ar, no endereço <http://localhost:9000>.

Figura 3 – Executando a aplicação Web

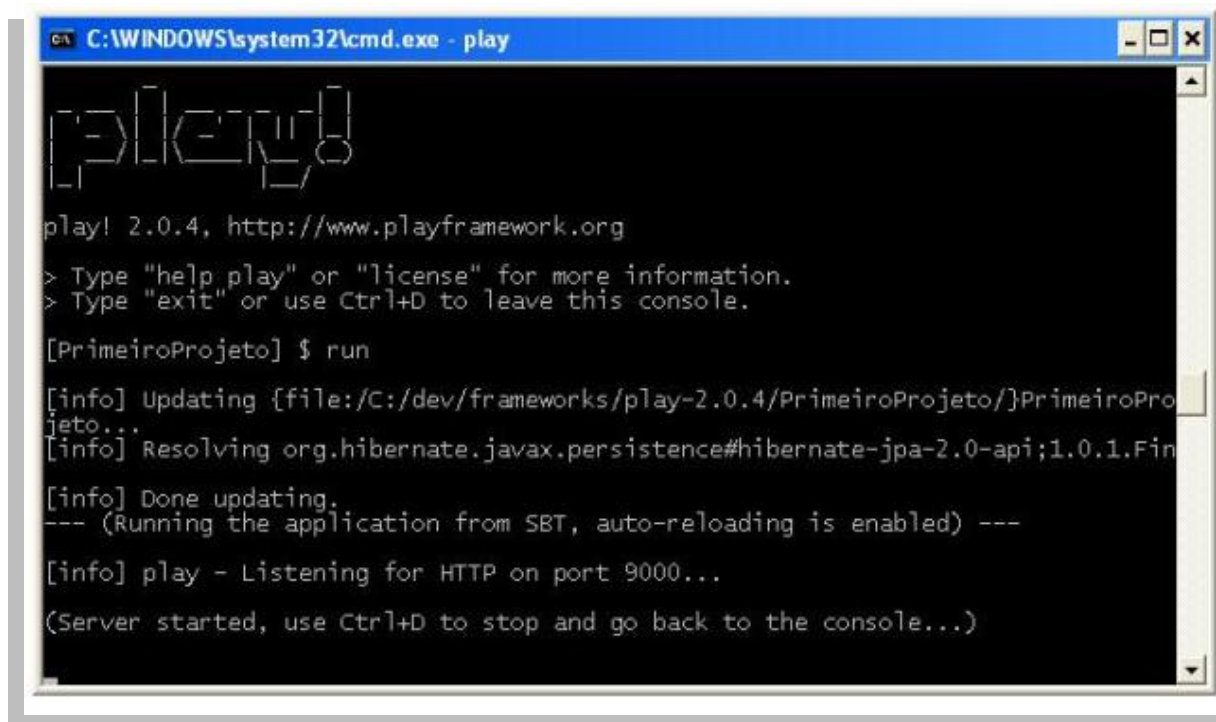
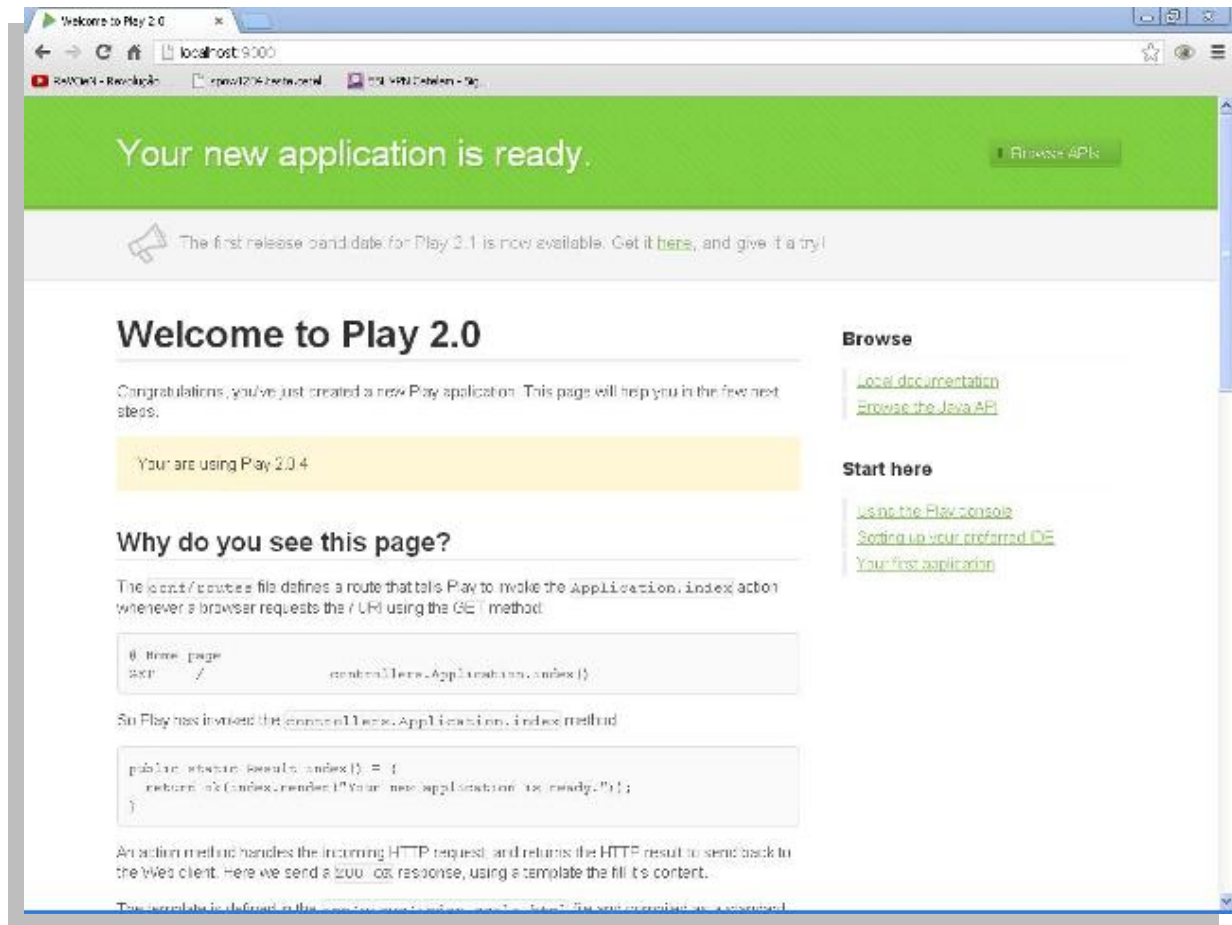


Figura 4 – Acessando a aplicação Web



Incrivelmente rápido, não? Agora vamos entender a mágica por trás dessa aplicação (O Play! Framework possui um servidor web embutido, o **JBoss Netty**).

Ao acessar a URL <http://localhost:9000>, o Play! vai consultar o arquivo **routes** (da pasta conf), para saber qual ação a ser efetuada.

Arquivo routes default:

```
# Routes
# Home page
GET      /                               controllers.Application.index()
# Map static resources from the /public folder to the /assets URL path
GET      /assets/*file                 controllers.Assets.at(path="/public", file)
```

O formato do arquivo routes é: HTTP Method | Request Path | action.

Como visto anteriormente, a linha "GET / controllers.Application.index()" nos diz então que ao chegar uma requisição GET na URI raiz da aplicação, deverá ser invocado o método estático index, da classe Application, do pacote (diretório) controllers.

Então, se verificarmos o diretório (*app/controllers*), veremos que o Play! criou uma classe Application com um método estático index. Simples assim. Seguindo a convenção, para adicionar outra ação, tipo /helloWorld, devemos:

- ✓ Adicionar uma nova linha em routes: GET /helloWorld controllers.Application.helloWorld();
- ✓ Adicionar um método estático helloWorld à classe Application;
- ✓ Acessar a URL: *http://localhost:9090/helloWorld*.

O mapeamento de URI's com as ações da classe é extremamente rápido, graças ao uso de convenções.

Analisaremos agora a classe **Application.java**, criada automaticamente pelo Play!

```
package controllers;
import play.*;
import play.mvc.*;
import views.html.*;
public class Application extends Controller {
    public static Result index() {
        return ok(index.render("Your new application is ready.));}
    // Método incluído para o exemplo anterior funcionar
    public static Result helloWorld() {
        return ok(index.render("Hello World!));}
    }
}
```

Já vimos que o método `index` e `helloWorld` são referenciados no arquivo `routes`. São métodos estáticos que devem retornar `Result`, que é uma interface que representa os códigos de status padrão do protocolo HTTP.

Assim, o método estático `ok` da classe `Results` (com `s`), representa o status 200 OK que será setado no cabeçalho de resposta do HTTP, o que significa que o cliente irá receber o conteúdo passado como argumento do método (`ok(contéudo)`). A classe `Results` mapeia todos os tipos de status HTTP, como `BAD Request(400)`, `FORBIDDEN (403)`, etc.

E quanto à `index`? Se procurarmos por algum fonte chamado `index.java`, não acharemos. O `index` na verdade se refere à página `index.scala.html`, da pasta `views`.

A questão agora é que podemos acessar qualquer página html dentro do código seguindo as seguintes convenções:

- A página deve estar no diretório `app\views`;
- Deve ter a seguinte nomenclatura: `<nome>.scala.html`.

Nota: não se preocupe com o `scala` no nome. É só uma convenção de nome.

Alterando método `helloWord`:

```
public static Result helloWorld() {  
    return ok(pagina.render("Hello World!"));  
}
```

Agora adicione a página html, de nome `pagina.scala.html` na pasta `app\views`.

HTML `pagina.scala.html`

```
@(message: String)  
  
<h1>@message</h1>
```


Acessando a URL: *http://localhost:9090/helloWorld*:
(Não é necessário reiniciar o Play!).

Figura 5 – Saída do hello Word



De fato, "acessamos" a página html definida na pasta views dentro do nosso código. Obviamente, o Play! Framework transforma a página html em código para que nosso Controller tenha acesso a ela. De novo o poder da convenção. Podemos trabalhar diretamente com código HTML (com Scala sendo a linguagem utilizada dentro das páginas HTML) e o Play! irá converter automaticamente para código sem que tenhamos que fazer nenhuma configuração, basta seguir a convenção.

Muitos podem ficar preocupados por ter que usar Scala como Expression Language dentro do template HTML. Mas a notação dessa EL é tão simples, que mesmo quem não programa em Scala irá assimilar rapidamente seu uso.

No caso do HTML *pagina.scala.html*

- **@(message: String)** - Indica que iremos receber um parâmetro do tipo String e que o mesmo será armazenado em message
- **<h1>@message</h1>** - Acesso a variável message.

Passamos parâmetros para a página através dos argumentos do método render. O retorno do método render, é o conteúdo da página processada, que será retornado para o cliente Web.

A ordem dos parâmetros é importante. Por exemplo:

Dois parâmetros passados para a página

```
public static Result helloWorld() {
    return ok(pagina.render("Hello World!", 1));
}
```

Alteração no HTML

```
@(message1: String, message2: Integer)
<h1>@message1</h1>
<h1>@message2</h1>
```

Retorno do método render
(que será enviado ao cliente Web)

```
<h1>Hello World!</h1>
<h1>1</h1>
```

Fácil não?

Quando chamamos a URI /, vemos uma página bem trabalhada, que possui, além da nossa mensagem enviada, conteúdo institucional do Play!

Página index.scala.html

```
@(message: String)
@main("Welcome to Play 2.0") {
    @play20.welcome(message, style = "Java")
}
```

A instrução `@(message: String)` declara uma variável `message` do tipo `String` (o argumento do método `render`).

A instrução `@main` indica que devemos acessar a página `main.scala.html`, que possui o seguinte conteúdo:

Página main.scala.html

```
@(title: String)(content: Html)
```

```
<title>@title</title>
```

```
<link rel="stylesheet" media="screen" href="@routes.Assets.at(" stylesheets=" "
main.css")"="">
```

```
<link rel="shortcut icon" type="image/png" href="@routes.Assets.at(" images=" "
favicon.png")"="">
```

```
<script src="@routes.Assets.at(" javascripts=" " jquery-1.7.1.min.js")"=""
type="text/javascript"></script>
```

```
@content
```

A página *main.scala.html* define duas variáveis, *title* e *content*. O valor de *title* é "Welcome to Play 2.0", enquanto *content* é o valor `@play20.welcome(message, style = "Java")`.

A linha: `@play20.welcome(message, style = "Java")` significa que estamos acessando a página *welcome.scala.html* da pasta `<diretório_de_instalação>\framework\src\play\src\main\scala\views\play20`. Podemos fazer uma analogia do `@play20` como um plugin, sendo acessada desse diretório.

Para ficar mais claro, vamos às convenções:

Todo diretório que for criado dentro dessa pasta:

```
<diretório_de_instalação>\framework\src\play\src\main\scala\views
```

É acessível pela anotação `@diretório.página`.

No caso, temos uma pasta *play20* em *views*:

```
<diretório_de_instalação>\framework\src\play\src\main\scala\views\play20
```

Que possui uma página chamada *welcome.scala.html*.

Para criar novos "plugins/templates", basta respeitar a lei da convenção.

6.3. DATABASE

O Play Framework fornece um plug-in para o gerenciamento de conexão JDBC. Você pode configurar quantos bancos de dados forem precisos.

Configurando JDBC Driver:

O Play é fornecido apenas com um driver de banco de dados **H2** na memória, útil para o modo desenvolvimento (ver figura *Play H2-Browser*). Consequentemente, para implantar na produção será necessário adicionar o driver de banco de dados como uma dependência.

Play H2-Browser

Importando Database:

Por exemplo, se você usa MySQL5, você precisará adicionar uma [[dependência | SBTDependencies]] para o conector:

```
Build.scala
val appDependencies = Seq(
  #javaJdbc
  "mysql" % "mysql-connector-java" % "5.1.21"
)
```

Propriedades da conexão com o Banco de Dados

Em seguida, você deve configurar a conexão no arquivo *conf / application.conf*. Por convenção, a fonte de dados JDBC padrão deve ser chamada *default*.

```
# Default database configuration  
db.default.driver=org.h2.Driver  
db.default.url="jdbc:h2:mem:play"
```

ou

```
# To configure MySQL  
db.default.url=mysql://localhost:root@secret/myDatabase
```

Acessando a fonte de dados JDBC

O pacote **play.db** fornece acesso às fontes de dados configuradas:

```
import play.db.*;  
  
DataSource ds = DB.getDataSource();
```

Obtendo a conexão JDBC:

Você pode recuperar uma conexão JDBC da seguinte maneira:

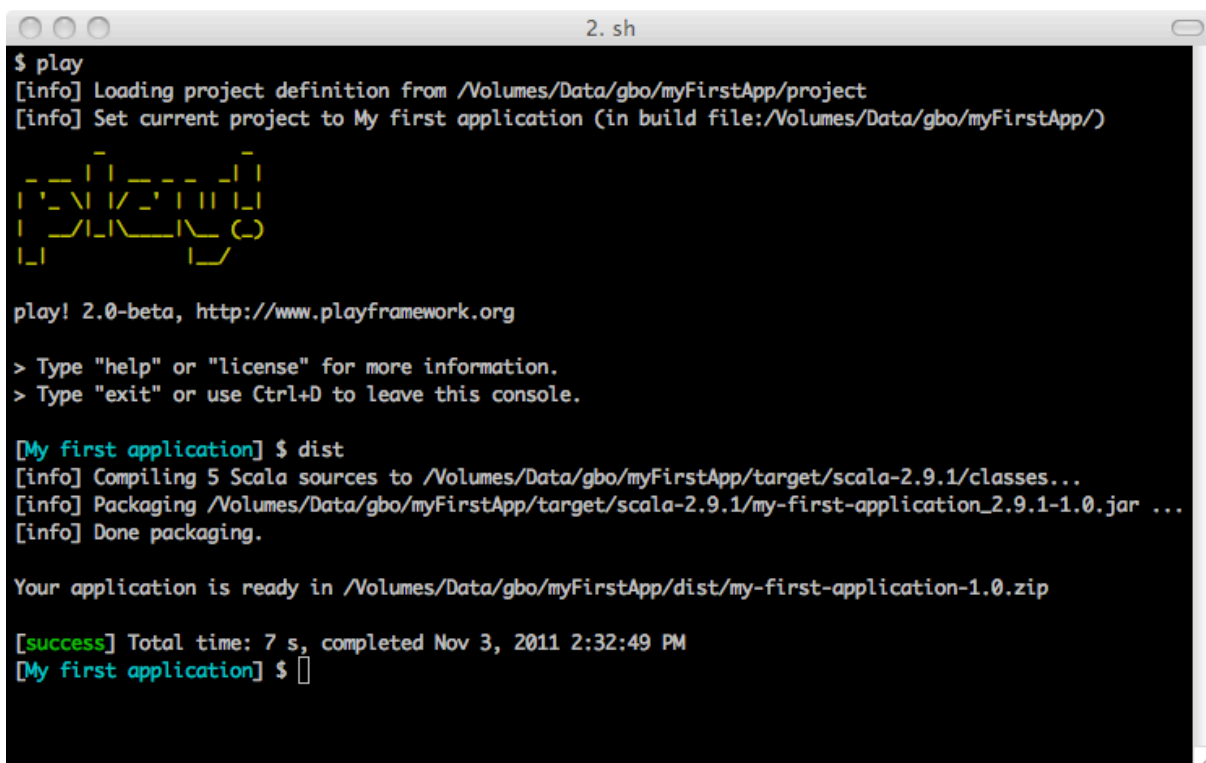
```
Connection connection = DB.getConnection();
```

É importante notar que as conexões resultantes não são automaticamente eliminadas no final do ciclo pedido. Em outras palavras, você é responsável por chamar o método **close()** em algum lugar no seu código, para que possam ser devidamente encerradas.

6.4. GERAR ARQUIVO JAR

Para construir uma versão binária do seu aplicativo e implantá-lo no servidor, sem qualquer dependência do Play em si, você precisará do comando **DIST**.

```
[PrimeiroProjeto] $ dist
```



```

2. sh
$ play
[info] Loading project definition from /Volumes/Data/gbo/myFirstApp/project
[info] Set current project to My first application (in build file:/Volumes/Data/gbo/myFirstApp/)

  _ _ _ _ _
 |'_\|/_\|'_\|
 | _\|/_\|/_\|
 | _\|/_\|/_\|
 | _\|/_\|/_\|

play! 2.0-beta, http://www.playframework.org

> Type "help" or "license" for more information.
> Type "exit" or use Ctrl+D to leave this console.

[My first application] $ dist
[info] Compiling 5 Scala sources to /Volumes/Data/gbo/myFirstApp/target/scala-2.9.1/classes...
[info] Packaging /Volumes/Data/gbo/myFirstApp/target/scala-2.9.1/my-first-application_2.9.1-1.0.jar ...
[info] Done packaging.

Your application is ready in /Volumes/Data/gbo/myFirstApp/dist/my-first-application-1.0.zip

[success] Total time: 7 s, completed Nov 3, 2011 2:32:49 PM
[My first application] $

```

Isso produz um arquivo ZIP contendo todos os arquivos JAR necessários para executar o aplicativo na pasta de destino de sua aplicação. O conteúdo do arquivo ZIP é organizado como:

```
[PrimeiroProjeto]
```

```

└ lib
  └ *.jar
    └ start

```

Você pode usar o comando **START** para executar o aplicativo.

Outra maneira é executar o comando **PLAY DIST** diretamente na janela de comandos do sistema operacional.

```
$ play dist
```

6.5. PUBLICAR

Você também pode publicar seu aplicativo para um repositório 'Maven'. A publicação irá enviar o arquivo JAR que contém o aplicativo e o arquivo POM (Project Object Model) correspondente.

Para isso, você terá que configurar o repositório que deseja publicar, no arquivo *project/Build.scala*.

```
val main = PlayProject(appName, appVersion, appDependencies).settings(
  publishTo := Some(
    "My resolver" at "http://mycompany.com/repo"
  )
  credentials += Credentials(
    "Repo", "http://mycompany.com/repo", "admin", "admin123"
  )
)
```

Então, no Play Console, use o comando **PUBLISH**.

```
[PrimeiroProjeto] $ publish
```

Outra opção, adotada para publicar a aplicação disponível neste trabalho, é a hospedagem na nuvem, através da plataforma de Cloud Computing **Heroku**.

Os serviços do Heroku para pequenas aplicações Java Web, bem como sites estáticos HTML, são gratuitos, possuem base de dados no PostgreSQL com limitação à 10 mil linhas.

Passos para o Deploy no Heroku:

A aplicação deve conter um arquivo em sua raiz, com o conteúdo:

```
web: target/start -Dhttp.port=${PORT} ${JAVA_OPTS}
```

Cadastrar-se e instalar o **Heroku Toolbelt**, disponível em:

<https://toolbelt.heroku.com/>

Através do console “**Git Bash**” digitar os comandos:

```
cd [Diretório da Aplicação]
git remote - v [Verificar o repositório da aplicação na nuvem]
git init
git add .
git commit -m "init"
git push heroku master [Publicar a aplicação no repositório 'heroku']
```

Caso haja Banco de Dados, **PostgreSQL**, adicionar dependências no arquivo: *project/Build.scala* :

Add Dependencies

```
"postgresql" % "postgresql" % "9.1-901.jdbc4"
```

Efetuando login no site do Heroku, acessar a base de dados para obter os dados de conexão da aplicação com o SGDB.

Exemplo:

Host	ec2-54-227-255-156.compute-1.amazonaws.com
Database	d9olrs6nq74jqs
User	rcyixjmjsknfrg
Port	5432
Password	4X8tlP2vlnomqhx8DmpvuuhE7x

Para desabilitar a segurança SSL (Transport Layer Security), caso não queira configurar o certificado digital, adicionar ao final da URL de conexão:

?sslfactory=org.postgresql.ssl.NonValidatingFactory

Na aplicação, abrir conexão com Banco de Dados:

```
private static Connection abrirConexao() throws SQLException{  
  
String url = "jdbc:postgresql://ec2-54-227-255-156.compute-  
1.amazonaws.com:5432/d9olrs6nq74jqs?sslfactory=org.postgresql.ssl.NonValidatingFactory";  
  
Properties props = new Properties();  
props.setProperty("user", "rcyixjmjsknfrg");  
props.setProperty("password", "4X8tIP2vlnomqhx8DmpvuuhE7x");  
props.setProperty("ssl", "true");  
Connection conn = DriverManager.getConnection(url, props);  
  
return conn;  
  
}
```

Para criar as tabelas e relacionamentos, realizar o download do SGDB PostgreSQL, disponível em:

<http://www.postgresql.org/download/>

A conexão com a base de dados remota, através do SGDB, pode ser feita diretamente utilizando os dados de conexão coletados anteriormente.

Os comandos de criação, bem como os de INSERT, SELECT, UPDATE e DELETE, etc., podem ser executados no **SQL Shell PSQL** provido pela instalação do SGDB.

7. CONCLUSÃO

✓ O Play Framework torna o desenvolvimento de aplicações Java e Scala uma tarefa fácil para o desenvolvedor.

✓ Como uma alternativa limpa e leve para as atuais aplicações Java Enterprise, o Play é focado na produtividade do desenvolvedor e tem por alvo a arquitetura RESTful que garante simplicidade e produtividade..

✓ O Play Framework cumpre o que promete, ao conseguir livrar o desenvolvedor de tarefas tediosas como configurar dezenas de arquivos de mapeamento, XML, etc., tornando o desenvolvimento Web divertido e produtivo, como a muito tempo não se via em Java.

✓ O Play possui uma série de módulos interessantes, que facilitarão ainda mais o desenvolvimento.

✓ Quem experimenta o Play Framework dificilmente irá ver os frameworks tradicionais MVC de Java do mesmo jeito.

✓ Nota: Este trabalho é embasado na documentação da versão **2.0** do Play Framework, podendo haver nuances ou variações ligeiras em relação a outras versões.

8. LINKS ÚTEIS

8.1. LINKS GERAIS

Documentação Play Framework 2.1	http://www.playframework.com/documentation/2.1.1/Home
Repositório de Módulos	http://www.playframework.com/modules
Templates	http://www.playframework.com/documentation/2.0.4/JavaTemplates
Utilização de IDE com Play Framework	http://www.playframework.com/documentation/1.2.4/ide
Vídeo: Play Framework 2.0 on Eclipse	http://www.youtube.com/watch?v=QaJI_nnSykU
Vídeo: Send Email	http://www.youtube.com/watch?v=qPUJ63chopA
Vídeo: Eclipse Debugger com Play	http://www.youtube.com/watch?v=1SCTOI0qrIM
Utilização de Banco de Dados com Play e JPA	http://www.slideshare.net/sedir/material-play-framework-uern-aula-03
Informações sobre REST	http://www.devmedia.com.br/boas-praticas-com-web-services-restful-java-magazine-83/18020
Fórum de Play Framework do Google	https://groups.google.com/forum/?fromgroups#!forum/play-framework

8.2. LINKS HEROKU: CLOUD APPLICATION PLATFORM

Heroku - Cloud Application Platform	https://www.heroku.com/
Publicar aplicação no Heroku	http://www.playframework.com/documentation/2.0/ProductionHeroku
Heroku Play Framework Support	https://devcenter.heroku.com/articles/play-support
Heroku PostgreSQL	https://devcenter.heroku.com/articles/heroku-postgresql

8.3. LINK DA APLICAÇÃO DESENVOLVIDA

Aplicação desenvolvida em Java, para Web, com utilização do Play Framework 2.0, Banco de Dados PostgreSQL e Heroku Cloud Application Platform.

<http://webagenda.herokuapp.com/>

Mais informações sobre a aplicação consulte a *Documentação da Aplicação*.

9. REFERÊNCIAS

- Play Framework, disponível em: <http://www.playframework.com/>. Acesso em: 15 de maio de 2013.
- Daniel Schmitz, Redescobrimo Java com Play! Framework, disponível em: <http://imasters.com.br/artigo/23777/java/redescobrimo-java-com-play-framework/>. Categoria: Artigo. Acesso em 15 de maio de 2013.
- DevMedia - asp.net, Java, Delphi e web Design, tudo em um só lugar! , disponível em: <http://www.devmedia.com.br/conhecendo-o-play-framework-para-java/26655>. Categoria: Artigo. Acesso em: 15 de maio de 2013.
- Erko Bridee: Uma visão particular de assuntos atuais..., disponível em: <http://blog.erkobridee.com/2011/12/05/play-framework-alta-produtividade-em-java/>. Acesso em: 15 de maio de 2013.
- Heroku – Cloud Application Platform: Play Framework Support, disponível em: <https://devcenter.heroku.com/articles/play-support>. Acesso em 22 de maio de 2013.