

Play Framework

Java para web sem servlets
e com diversão

play 

Agradecimentos

Agradeço a você por querer aprender mais, à minha esposa por sempre estar ao meu lado, aos meus pais e a Deus por tudo.

E segue o jogo!

Sobre o autor

Formado pela UNESP em BCC, foi instrutor oficial da Sun Microsystems e da Oracle Education. Atualmente contribui para alguns projetos open source, como KDE, Jenkins entre outros.

Prefácio

O melhor presente que se dá é aquele que você gostaria de ganhar.

Esse é o livro que eu gostaria de ler quando estava começando a usar o Play Framework, ele é o meu presente para você, aproveite!

Público alvo

Esse livro foi feito para programadores Java (iniciantes e veteranos) que buscam pelo desenvolvimento rápido e divertido de aplicações web.

Quickstart – a primeira parte do livro

Para rapidamente configurar o seu ambiente de desenvolvimento, disponibilizar a sua aplicação bonita, acessando banco de dados e publicando na nuvem não será preciso ler todos os capítulos, apenas os quatro primeiros.

Melhorando sua aplicação – a segunda parte do livro

Os capítulos restantes complementam a sua aplicação com a criação de serviços, autenticação e o uso de alguns plugins imperdíveis para o seu sistema.

Código fonte

O código fonte desse livro está disponível no endereço <https://github.com/boaglio/play2-casadocodigo>, onde foram criadas tags para cada um dos capítulos, para facilitar a compreensão da evolução do nosso sistema de filmes cult.

Sumário

1	Hello Play Java e Hello Play Scala	1
1.1	O que é o Play	1
1.2	O que não é o Play	1
1.3	Instalação do Play	3
1.4	Sua primeira aplicação Java	6
1.5	A estrutura da aplicação Play	7
1.6	Subindo sua aplicação Play	8
1.7	Olá Scala	9
1.8	Preciso saber Scala?	12
1.9	Próximos passos	12
2	Navegando com estilo	15
2.1	Bem-vindo ao Eclipse	15
2.2	Navegação	20
2.3	Adicionando estilo	22
2.4	Organizando as páginas	25
2.5	Próximos passos	28
3	Persistindo seus dados	29
3.1	Próximos passos	44
4	Publicando em qualquer lugar	45
4.1	Rede local	45
4.2	Servidor Java EE	46
4.3	Deploy na nuvem	46
4.4	Próximos passos	54

5	Melhorando o input do usuário	55
5.1	Isolando mensagens	59
5.2	Tratando erros	62
5.3	Páginas customizadas	63
5.4	Próximos passos	65
6	Criando serviços	67
6.1	Acessando serviço via web	67
6.2	Debugando pelo Eclipse	69
6.3	Acessando serviço via mobile	71
6.4	Próximos passos	73
7	Integrando nas redes sociais	75
7.1	Criando uma aplicação no Facebook	75
7.2	Integração via JavaScript	77
7.3	Integração via SecureSocial	80
7.4	Próximos passos	87
8	Melhorias na aplicação	89
8.1	Configurando HTTPS	89
8.2	Lendo constantes globais	93
8.3	Upload de imagem	93
8.4	Testando sua aplicação	102
8.5	Próximos passos	113
9	Continue seus estudos	115
9.1	Para saber mais	115
10	Apêndice A - Play console em detalhes	117
11	Apêndice B - Instalação e configuração do PostgreSQL	119
12	Apêndice C - Instalação e configuração do Android	129

CAPÍTULO 1

Hello Play Java e Hello Play Scala

1.1 O QUE É O PLAY

O Play é um framework que redefine o desenvolvimento web em Java. O seu foco é o divertido desenvolvimento no qual a interface HTTP é algo simples, flexível e poderoso, sendo uma alternativa limpa para as opções Enterprise Java infladas. Ele foca na produtividade do desenvolvedor para as arquiteturas RESTful, e sua vantagem em relação às linguagens e frameworks não Java, como Rails e PHP, é que ele usufrui de todo o poder da Java Virtual Machine (JVM).

1.2 O QUE NÃO É O PLAY

O Play não é um framework padrão Java EE, como Spring, Struts ou VRaptor – ele usa uma arquitetura extremamente simples.

Uma aplicação JSF roda sobre a API de Servlet, que por sua vez roda em um container Java EE, que fica dentro de um HTTP Server.

Perceba que todo desenvolvedor é obrigado a trabalhar com essas quatro camadas. Já com o Play, temos apenas duas: o próprio Play framework e o seu HTTP server embutido (Netty).

Além disso, ele não é apenas um framework web, ele é uma solução completa que envolve persistência e muito mais recursos, como:

- Servidor HTTP integrado;
- Acesso completo à interface HTTP;
- API de serviços REST;
- Código cujas mudanças é possível testar com um simples reload de página;
- Engine de template de alta performance;
- Permitir que sua aplicação funcione para vários idiomas;
- Compilação dos arquivos estáticos de sua aplicação;
- Suporte a I/O assíncrono;
- Validação HTML do input do usuário;
- Cache integrado;
- Sistema de build próprio e integrado (sbt);
- Plataforma akka criada para ambiente distribuído e à prova de falhas;
- Persistência de dados.

Confira a visão geral do Play na figura [1.1](#)

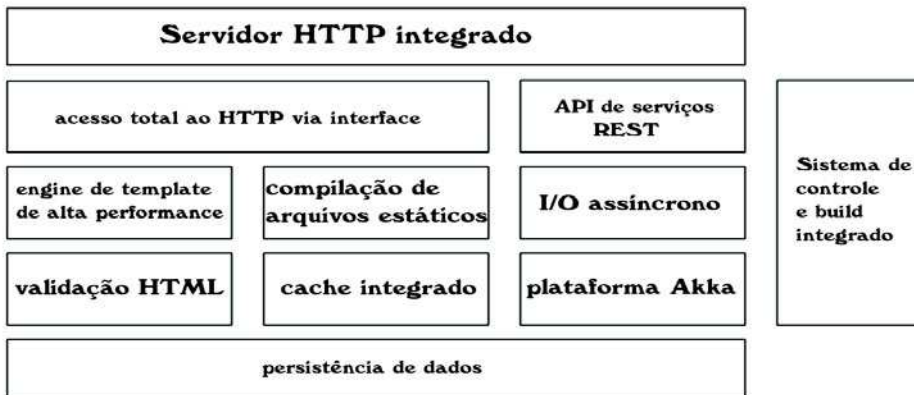


Figura 1.1: Play framework stack

1.3 INSTALAÇÃO DO PLAY

A instalação do Play é bem simples e feita em apenas dois passos. O primeiro deles é fazer o download do site <http://www.playframework.org>. Depois disso, faça o ajuste conforme o seu sistema operacional.

Instalação no Windows

Descompacte o pacote na raiz e renomeie o diretório compactado para `play`. Exemplo: o arquivo `play-2.2.1.zip` criará o diretório `C:\play-2.2.1\`. Renomeie-o para `C:\play\`.

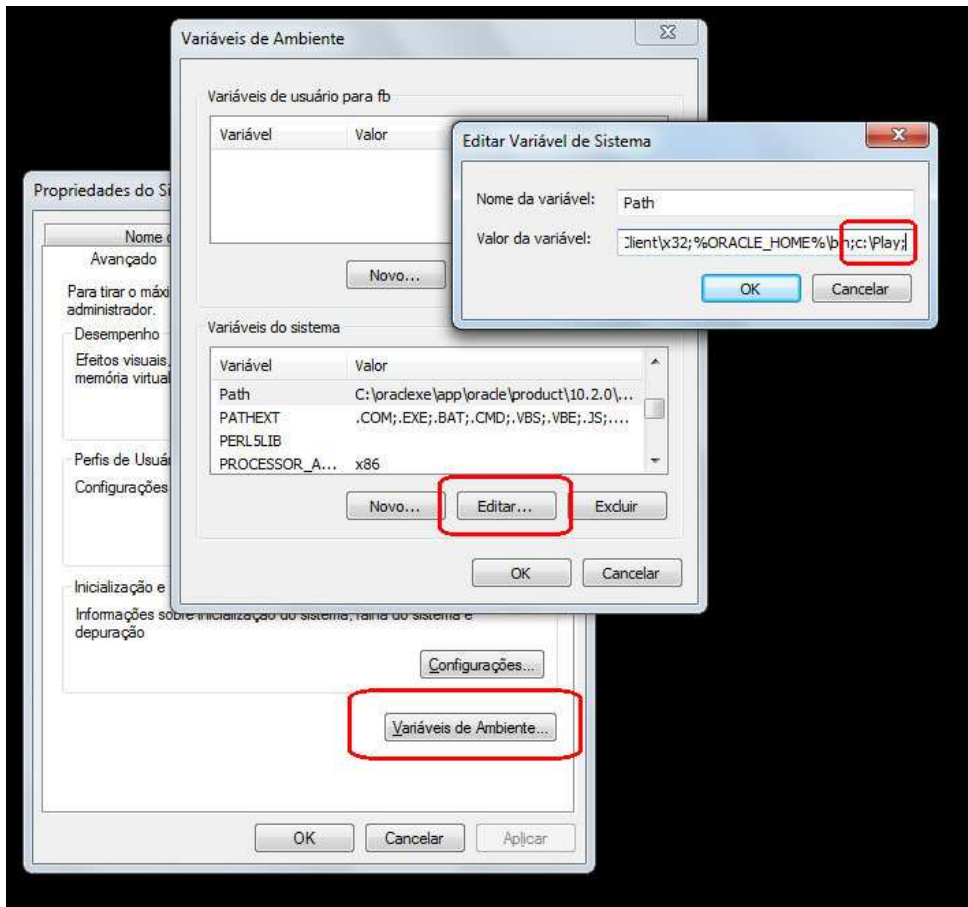


Figura 1.2: Variável de ambiente no Windows

Edite a variável `PATH` conforme a figura 1.2, adicionando no final do `PATH` o valor de `;c:\play\`.

ATRÁS DE UM PROXY

Se sua internet estiver atrás de um proxy, altere o arquivo `C:\play\framework\build.bat` na linha de comando Java e adicione os parâmetros:

```
-Dhttp.proxyUser=<meu-usuario>
-Dhttp.proxyPassword=<minha-senha>
-Dhttp.proxyHost=<servidor>
-Dhttp.proxyPort=<porta>
```

Caso seja lançada uma atualização do Play, e você queira atualizar, renomeie o diretório `C:\play\` para `C:\play-old\` e descompacte a nova versão em `C:\play\`.

Instalação em Linux

Descompacte o pacote na raiz, por exemplo: `/home/fb/play-2.2.1/`, e crie um link simbólico para esse diretório chamado `play`, como:

```
ln -s /home/fb/play-2.2.1/ /home/fb/play
```

Caso haja uma atualização, descompacte a nova versão e atualize o link simbólico para o novo diretório.

Adicione no arquivo `$HOME/.bashrc` ou em `$HOME/.bash_profile` o comando: `export PATH=$PATH:$HOME/play/`.

Instalação em Mac OSX

Descompacte o pacote na raiz, por exemplo: `/home/fb/play-2.2.1/`, e crie um link simbólico para esse diretório chamado `play`, como:

```
ln -s /home/fb/play-2.2.1/ /home/fb/play
```

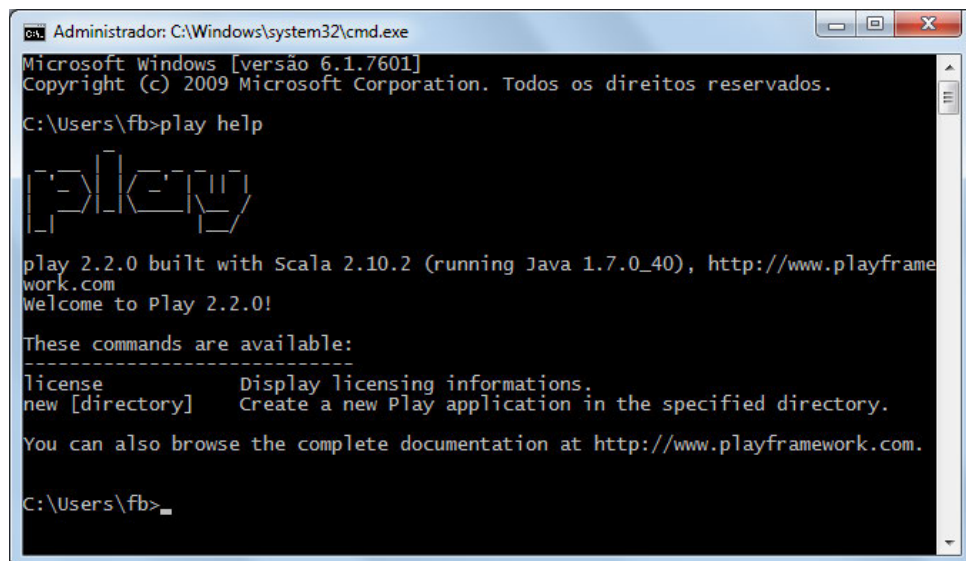
Adicione no arquivo `/etc/paths` o diretório `$HOME/play/`.

Em uma eventual atualização, descompacte a nova versão e atualize o link simbólico para o novo diretório.

Se preferir usar o Homebrew, apenas rode o comando `brew install play`.

Testando sua instalação

Depois de configurado, abra o console do seu sistema operacional e digite `play help`. O resultado esperado está na figura 1.3.



```

Administrador: C:\Windows\system32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\fb>play help

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

play 2.2.0 built with Scala 2.10.2 (running Java 1.7.0_40), http://www.playframework.com
Welcome to Play 2.2.0!

These commands are available:
-----
license           Display licensing informations.
new [directory]   Create a new Play application in the specified directory.

You can also browse the complete documentation at http://www.playframework.com.

C:\Users\fb>

```

Figura 1.3: Instalação do Play com sucesso no Windows

Pronto! Agora que o Play está instalado, podemos começar a nossa primeira aplicação!

1.4 SUA PRIMEIRA APLICAÇÃO JAVA

Para criar a nossa primeira aplicação, é só digitar `play new <nome-da-aplicação>`:

```
fb@cascao ~/workspace-play > play new play-java
```

```

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

```

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
```

```
http://www.playframework.com
```

The new application will be created in /home/fb/workspace-play/play-java

```
What is the application name? [play-java]
>
```

Depois escolha a opção 2, que vai criar uma aplicação Play Java:

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

- 1 - Create a simple Scala application
- 2 - Create a simple Java application

```
> 2
```

OK, application play-java is created.

Have fun!

```
fb@cascao ~/workspace-play >
```

E pronto, sua aplicação está criada!

1.5 A ESTRUTURA DA APLICAÇÃO PLAY

Vamos apenas olhar o que foi criado, sem muitos detalhes:

```
.
|-- app                                (arquivos Java da aplicação)
|   |-- controllers
|   |   |-- Application.java
|   |-- views
|       |-- index.scala.html
|       |-- main.scala.html
|-- build.sbt
|-- conf                                (arquivos de configuração)
|   |-- application.conf
|   |-- routes
|-- project
|   |-- build.properties
|   |-- plugins.sbt
```



```
|-- public                                (arquivos estáticos)
|   |-- images
|   |   `-- favicon.png
|   |-- javascripts
|   |   `-- jquery-1.9.0.min.js
|   `-- stylesheets
|       `-- main.css
|-- README
`-- test                                (arquivos para testes)
    |-- ApplicationTest.java
    `-- IntegrationTest.java
```

1.6 SUBINDO SUA APLICAÇÃO PLAY

A administração de sua aplicação é feita pelo `play console`, que possui diversos comandos de gerenciamento.

Para chamar o console, dentro do diretório criado na sua aplicação digite `play`, e depois o comando `run` para subir.

```
fb@cascao ~/workspace-play/play-java > play
[info] Loading project definition from
/home/fb/workspace-play/play-java/project
[info] Set current project to play-java (in build
file:/home/fb/workspace-play/play-java/)
```

```

      _
 _ _ _ | | _ _ _ _ _
| ' _ \ | / _ ' | | |
| __/|_| \____| \_ /
|_|          |__/_
```

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

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

```
[play-java] $ run
[info] Updating {file:/home/fb/workspace-play/play-java/}play-java...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
```

--- (Running the application from SBT, auto-reloading is enabled) ---

```
[info] play - Listening for HTTP on /0.0.0.0:9000
```

(Server started, use Ctrl+D to stop and go back to the console...)

Em seguida, é possível acessar a aplicação através do browser pelo endereço <http://localhost:9000/>, obtendo um resultado semelhante à figura 1.4.

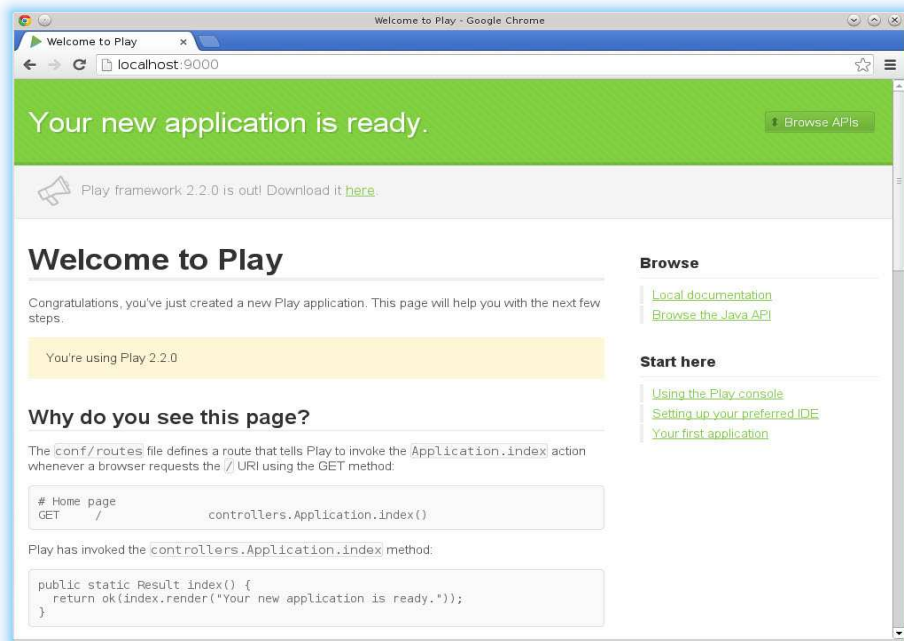


Figura 1.4: Rodando Play

Para derrubar o servidor, tecele `Control+D` e, para sair do `play console`, digite `exit`.

1.7 OLÁ SCALA

Durante o livro, vamos focar no Java, mas criar e rodar uma aplicação em Scala é parecido:

```
fb@cascao ~/workspace-play > play new play-scala
```

```

      _
    _-_-|_|_-_-_-_-
|'_-\| |/_-'| || |
|_--/|_| \----\ \_ /
|_|          |__/_/

```

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

The new application will be created in /home/fb/workspace-play/play-scala

```
What is the application name? [play-scala]
```

```
>
```

Depois selecione a opção 1, que vai criar uma aplicação Play Scala:

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

- 1 - Create a simple Scala application
- 2 - Create a simple Java application

```
> 1
```

```
OK, application play-scala is created.
```

Have fun!

```
fb@cascao ~/workspace-play > cd play-scala/
fb@cascao ~/workspace-play/play-scala > play
[info] Loading project definition from
/home/fb/workspace-play/play-scala/project
[info] Set current project to play-scala
(in build file:/home/fb/workspace-play/play-scala/)
```

```

      _
    _-_-|_|_-_-_-_-
|'_-\| |/_-'| || |
|_--/|_| \----\ \_ /
|_|          |__/_/

```

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

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

[play-scala] $ run
[info] Updating {file:/home/fb/workspace-play/play-scala/}play-scala...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.

--- (Running the application from SBT, auto-reloading is enabled) ---

[info] play - Listening for HTTP on /0.0.0.0:9000

(Server started, use Ctrl+D to stop and go back to the console...)

[info] Compiling 5 Scala sources and 1 Java source to
/home/fb/workspace-play/play-scala/target/scala-2.10/classes...
[info] play - Application started (Dev)
```

O resultado também é semelhante à figura 1.5.

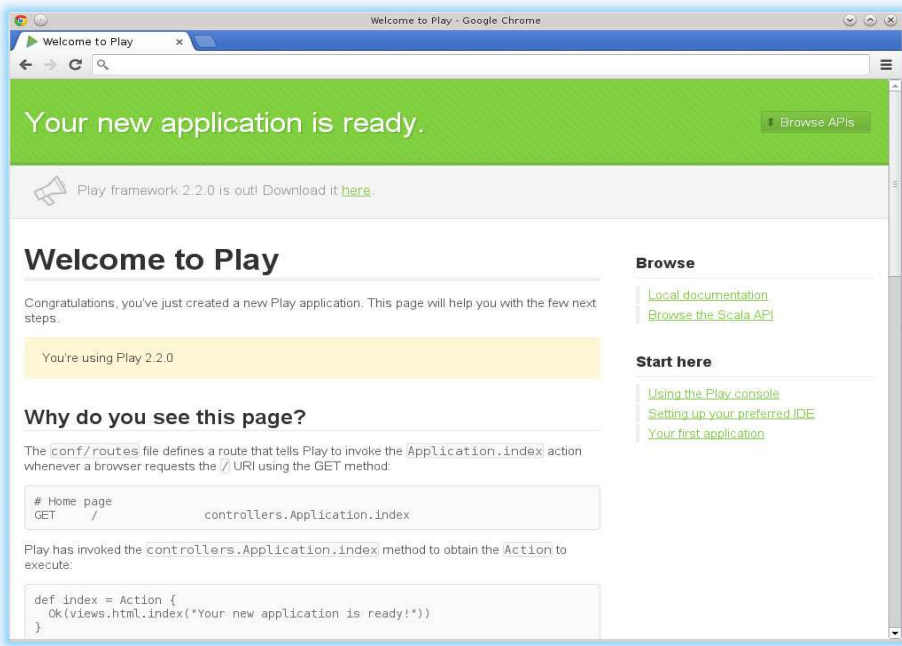


Figura 1.5: Rodando Play

1.8 PRECISO SABER SCALA?

Felizmente não. O core do Play 2 é feito em Scala, mas ele é perfeito para Java, pois podemos trabalhar com ele sem aprender uma nova linguagem e ambos usam a nossa querida JVM.

1.9 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- visão geral do Play Framework;
- como instalar o Play;
- como criar sua primeira aplicação em Java e Scala.

Agora que já molhamos os pés, nos próximos capítulos vamos aprender a nadar criando o primeiro CRUD.

CAPÍTULO 2

Navegando com estilo

A aplicação do Play possui uma estrutura simples que facilita muito o desenvolvimento de aplicações. Nesse capítulo, veremos como deixar sua aplicação com uma interface mais profissional e como controlar a navegação entre as telas usando os templates.

Mas antes disso, vamos sair do console e aprender a configurar sua aplicação no Eclipse?

2.1 BEM-VINDO AO ECLIPSE

Vamos criar uma aplicação chamada “filmes”, da mesma maneira que você viu no capítulo anterior.

- play new filmes
- cd filmes
- play

Acessando o console do Play, usamos o comando `eclipse` para preparar o seu projeto:

```
fb@cascao /home/fb/workspace-play/filmes > play
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
```

```

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

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

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

```
[filmes] $ eclipse
[info] About to create Eclipse project files for your project(s).
[info] Updating {file:/home/fb/workspace-play/filmes/}filmes...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 4 Scala sources and 2 Java sources to
/home/fb/workspace-play/filmes/target/scala-2.10/classes...
[info] Successfully created Eclipse project files for project(s):
[info] filmes
[filmes] $
```

Os arquivos necessários foram criados para importamos o projeto.

Abra o seu Eclipse no workspace em que criou o projeto e selecione a opção `Import` conforme indicado na figura:

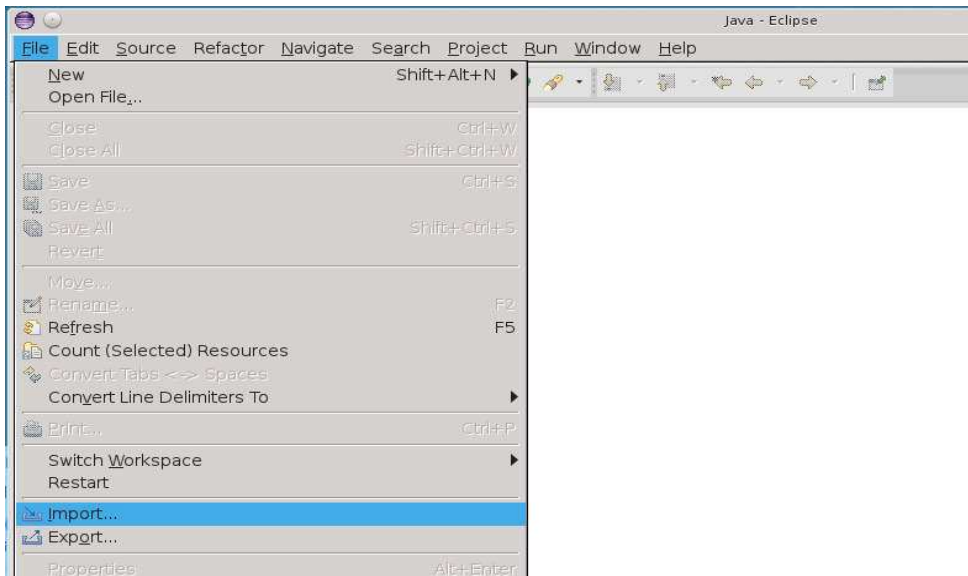


Figura 2.1: Importando o projeto com Eclipse

Em seguida, escolha a opção `Existing Projects into workspace`. Veja a imagem:

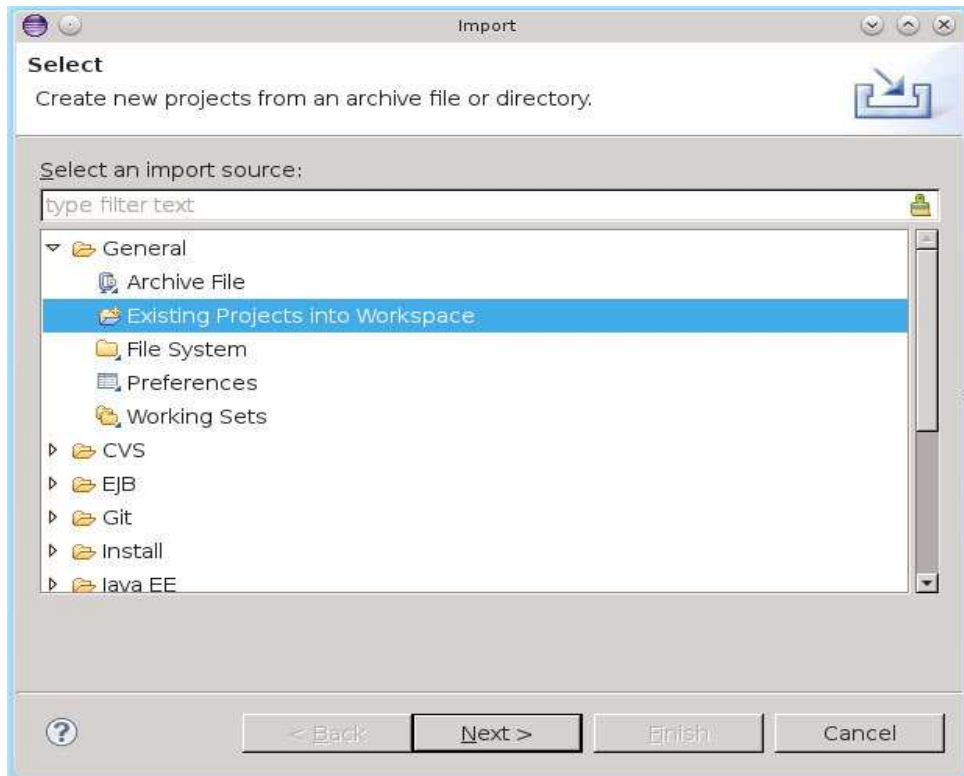


Figura 2.2: Escolha a opção de projetos existentes

Depois de indicar o diretório do seu workspace, escolha o projeto filmes e selecione `Finish` para concluir a importação:

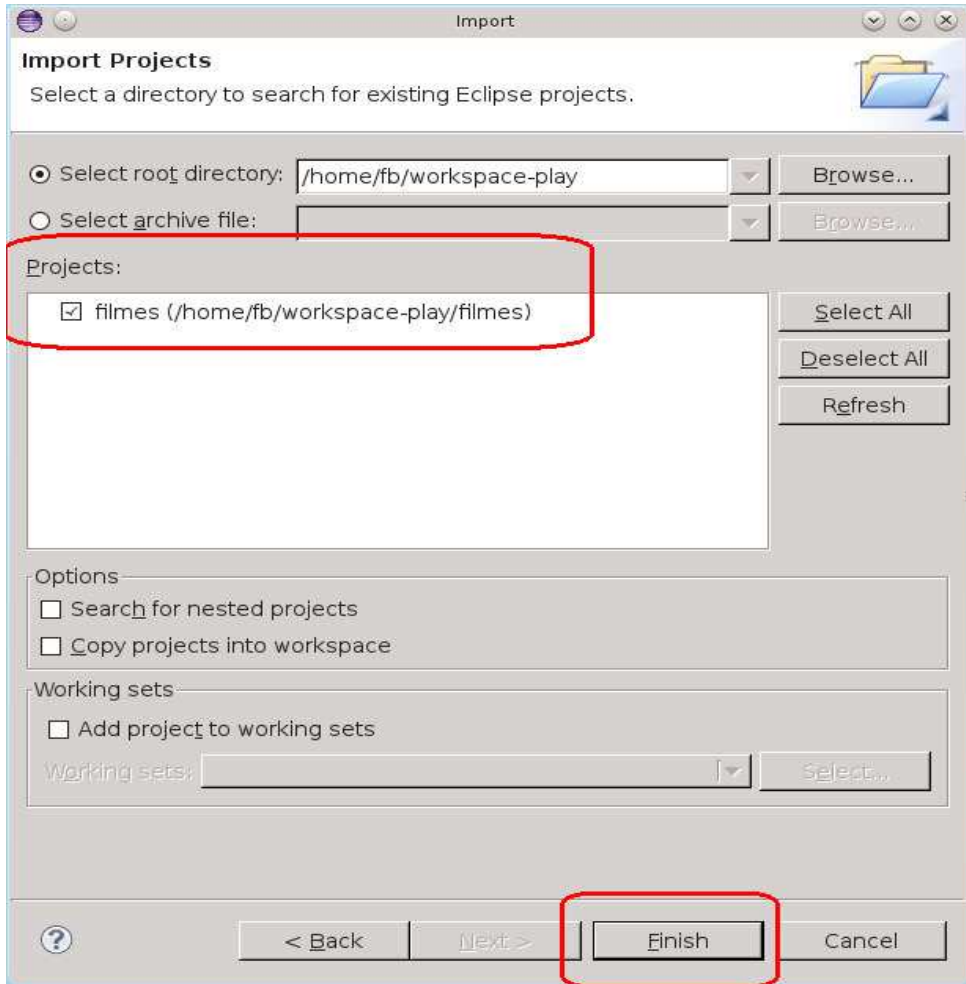


Figura 2.3: Finalize a importação do projeto

PARA OS USUÁRIOS DO INTELLIJ IDEA

Para preparar o seu projeto para essa IDE, use o comando `idea` dentro do console do Play.

Pronto, tudo está dentro do Eclipse, ficou mais fácil de trabalhar!

Agora que importamos o projeto, temos diversos arquivos organizados no nosso Eclipse. Precisamos entender para que serve cada um deles.

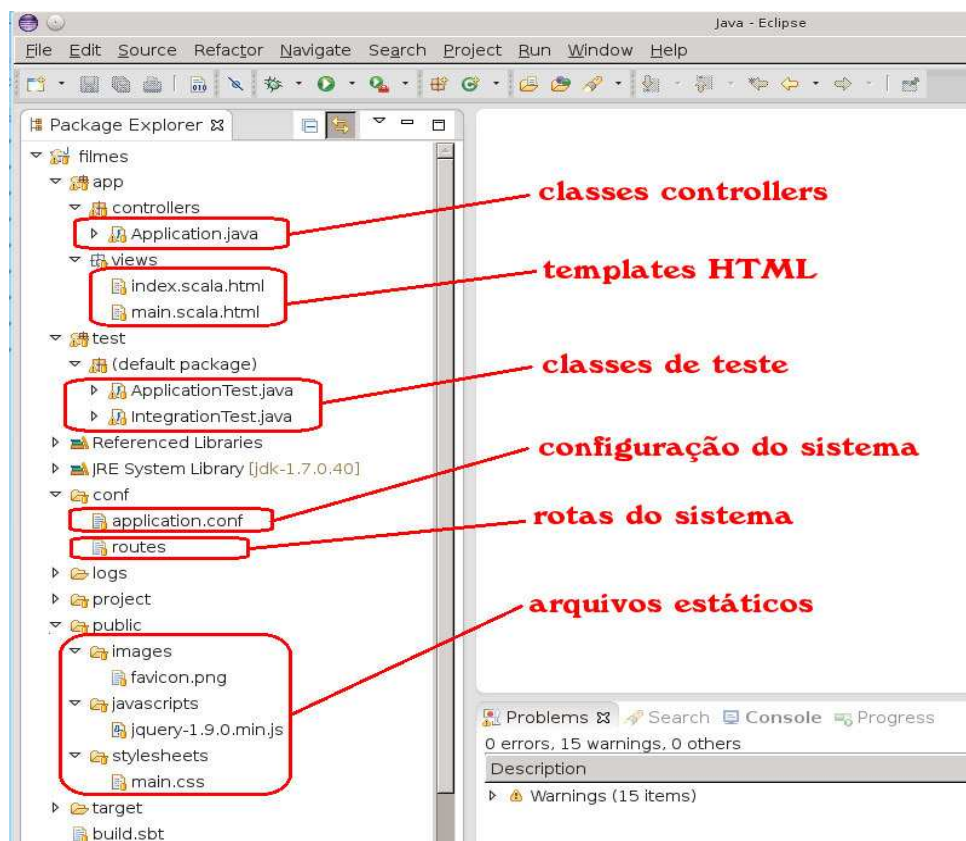


Figura 2.4: Arquivos do projeto filmes

2.2 NAVEGAÇÃO

Qual link chama qual método? Em aplicações Java EE esse mapeamento pode ser feito de diversas maneiras e em diferentes lugares, o que torna complicado o controle de alterações de um sistema. Felizmente, no Play tudo isso é concentrado em um único arquivo, que armazena o mapeamento de todas as rotas que uma aplicação suporta.

O arquivo de rotas, `conf/routes`, é dividido em três partes definidas nessa ordem:

- 1) método HTTP – GET ou POST;
- 2) URI da aplicação;

3) classe que será chamada ao acessar a URI com o método HTTP.

Por padrão, a página raiz vem com a configuração conforme a figura:

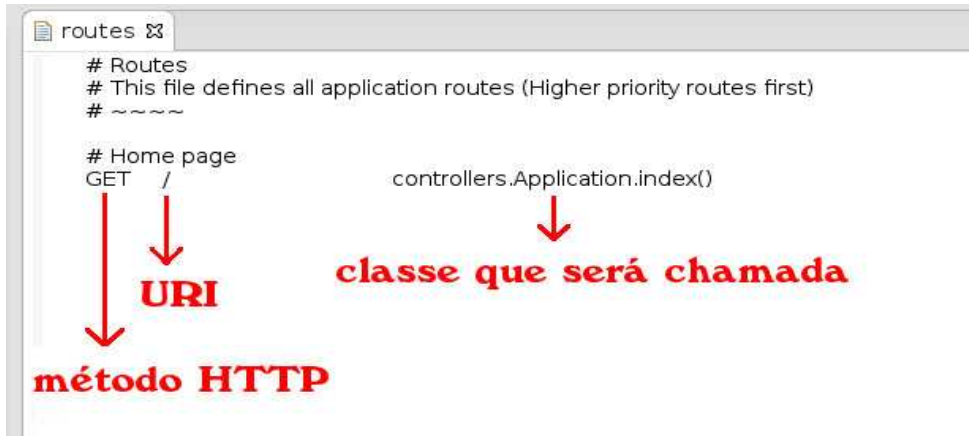


Figura 2.5: Arquivo de rotas

Vamos entender melhor o funcionamento desse arquivo criando uma página de informações sobre o sistema.

Escrevemos um método que tratará as informações na classe existente `Application`, que já contém o método `index` responsável pela página inicial.

```
public static Result sobre() {
    return ok("Sobre");
}
```

Todo método de um controller do Play deve seguir três regras simples:

- 1) deve ser public;
- 2) deve ser static;
- 3) deve retornar um objeto do tipo `Result` (ou de uma subclasse dele).

Fazendo o acesso à URL <http://localhost:9000/sobre> recebemos a mensagem de erro da figura 2.6, pois criamos o controlador, mas não mapeamos a rota.

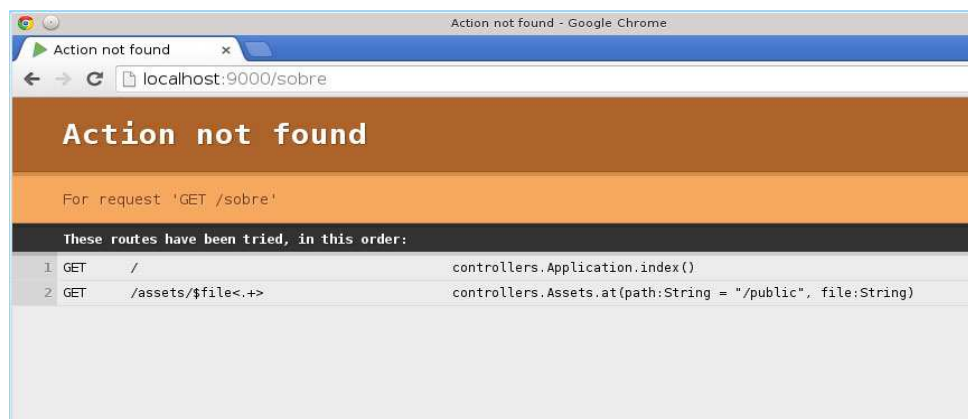


Figura 2.6: Erro ao acessar URL sem rota mapeada

Vamos mapear a rota da nossa nova página, adicionando ao arquivo `routes` a seguinte linha:

```
# método HTTP GET - URI - método que será chamado
GET                /sobre          controllers.Application.sobre()
```

Logo após adicionarmos essa linha, a página é exibida corretamente, apenas com um texto simples.



Figura 2.7: Página criada que exibe texto simples

2.3 ADICIONANDO ESTILO

Vamos deixar nossa aplicação mais elegante? Para isso, vamos aproveitar o excelente trabalho de Mark Otto e Jacob Thornton e usar o conjunto de templates chamado Bootstrap.

Começaremos fazendo o download do arquivo zipado em <https://github.com/twbs/bootstrap/archive/v3.0.0.zip> e descompactando-o em um diretório temporário (Exemplo: C:\TEMP\).

Em seguida, copiamos o conteúdo do diretório `dist` (C:\TEMP\bootstrap-3.0.0\dist\) para o diretório `public` da nossa aplicação `filmes`.

Vamos criar um template para a nossa página `Sobre` criando um arquivo dentro de `app.views` chamado `sobre.scala.html`, com o seguinte conteúdo:

```
@(sistema: String)(versaoDoPlay : String)
```

Aqui temos uma característica bem interessante do Play: conseguimos definir parâmetros para o nosso template – nesse caso, dois do tipo `String`.

Vamos usar o primeiro parâmetro, `sistema`, para definir o título (`title`) e a descrição (`h3`) da página.

```
<!DOCTYPE html>
<html>
  <head>
    <title>@sistema</title>
```

Podemos usar os estilos e scripts do Bootstrap e do Play.

```
<link href=
  "@routes.Assets.at("bootstrap/css/bootstrap.min.css")"
  rel="stylesheet" media="screen">
<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-1.9.0.min.js")"
  type="text/javascript">
</script>
</head>
```

Por fim, o restante do layout da página.

```
<body>
  <div class="container">
    <div class="header">
```



```

    <ul class="nav nav-pills pull-right">
      <li class="active"><a href="/">Home</a></li>
    </ul>
    <h3 class="text-muted">@sistema</h3>
  </div>
  <div class="jumbotron">
    <p><a class="btn btn-lg btn-success" href="#">
      powered by Play Framework @versaoDoPlay</a></p>
  </div>
</div>
</body>
</html>

```

Logo após a criação desse template, o Play compilou uma classe chamada `views.html.sobre`, que usaremos no nosso controller `Application` dessa maneira:

```

public static Result sobre() {
  // ok("Sobre");
  return ok(views.html.sobre.render(
    "Top 100 filmes Cult",
    play.core.PlayVersion.current()
  ));
}

```

Além de criar a classe `views.html.sobre`, foi criado também o método `render` com os dois parâmetros `String` que escolhemos.

Logo após gravar o controller, sem precisar fazer restart algum, faça apenas um reload na página e o resultado será o seguinte:



Figura 2.8: Página simples com estilos

Agora está melhor, não?

2.4 ORGANIZANDO AS PÁGINAS

Vamos continuar colocando um menu e reorganizando as páginas.

A página inicial será alterada, mas vamos antes analisar a que vem por padrão.

```
@(message: String)

@main("Welcome to Play") {

    @play20.welcome(message, style = "Java")

}
```

Além de essa página ter um parâmetro do tipo `String`, ela também chama outro template (o `main`) passando dois parâmetros: o primeiro deles do tipo `String` e o segundo (entre chaves) do tipo `HTML`.

A nova página inicial `inicio.scala.html` também chamará o template `main`, porém passando parâmetros diferentes:

```
@main("Top 100 filmes Cult") {

<style>
body {
    padding-top: 50px;
}
.starter-template {
    padding: 40px 15px;
    text-align: center;
}

</style>
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle"
                data-toggle="collapse" data-target=".navbar-collapse">
            </button>
            <a class="navbar-brand" href="/">Top 100 filmes Cult</a>
        </div>
    </div>
</div>
```

```

    </div>
    <div class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li><a href="/sobre">Sobre o sistema</a></li>
        <li><a href="/play">Sobre o Play Framework</a></li>
        <li>
          <a href="http://getbootstrap.com/">Sobre o Bootstrap</a>
        </li>
      </ul>
    </div>
  </div>
</div>
<div class="container">
  <div class="starter-template">
    <h1>Top 100 filmes Cult</h1>
    <p class="lead">O melhor do cinema est&aacute; aqui !</p>
  </div>
</div>
}

```

O nosso segundo passo é criar um novo controller para chamar o novo template:

```

package controllers;

import play.mvc.Controller;
import play.mvc.Result;

public class Inicio extends Controller {

    public static Result index() {
        return ok(views.html.inicio.render());
    }
}

```

E o último passo é, no arquivo de rotas, alterar a padrão para o novo controller:

```

#GET      /                               controllers.Application.index()
GET       /                               controllers.Inicio.index()

```

Fazendo um reload na página inicial temos como resultado a figura 2.9, mas com HTML bem simples.

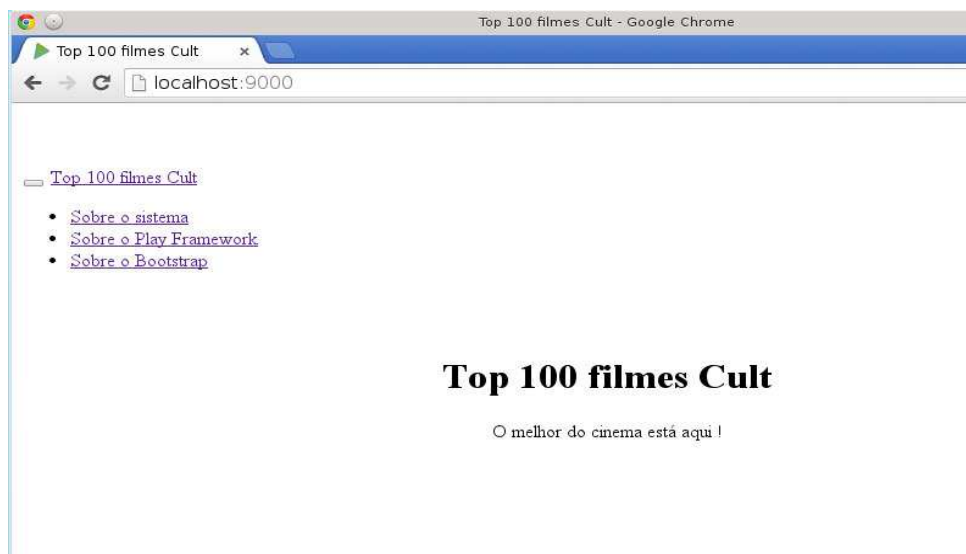


Figura 2.9: Nova página principal

O que falta é usar os estilos do Bootstrap dentro do template `main`, já que ele será utilizado em todas as páginas. Vamos adicionar uma linha chamando os estilos logo após a tag `title`:

```
<title>@title</title>
<link href="@routes.Assets.at("bootstrap/css/bootstrap.min.css")"
      rel="stylesheet"
      media="screen">
```

Fazendo um novo reload na página inicial temos como resultado com os estilos aplicados como na figura:



Figura 2.10: Nova página principal com estilos

Para completar, vamos adicionar a rota da antiga página inicial para o link “Sobre o Play”. O arquivo final ficou assim:

GET	/	controllers.Inicio.index()
GET	/assets/*file	controllers.Assets.at(path="/public", file)
GET	/sobre	controllers.Application.sobre()
GET	/play	controllers.Application.index()

2.5 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a configurar o Eclipse com uma aplicação Play;
- a estrutura de arquivos de uma aplicação;
- como alterar a rota de navegação de páginas;
- como aplicar estilos.

Agora que já temos uma aplicação navegável, nos próximos capítulos vamos criar o primeiro CRUD acessando banco de dados.

CAPÍTULO 3

Persistindo seus dados

Praticamente todos os principais sites do mercado usam banco de dados para manipular suas informações, e no nosso sistema não será diferente.

Nesse capítulo veremos com detalhes a facilidade de persistência de dados com Play através do cadastro de diretores de um filme, descrevendo desde a classe de domínio (Model), passando pelo Controller e terminando na página JSP (View).

O gerenciador objeto relacional do Play é chamado de **ebean**, que usa o padrão JPA. Sua implementação é conhecida e bem divulgada pela comunidade Java.

Vamos iniciar a parte de persistência de nosso sistema adicionando uma classe importante para o nosso negócio: *Diretor*.

Ela precisa ficar dentro do pacote `models` e estender a classe `play.db.ebean.Model`:

```
package models;

import javax.persistence.*;
import play.data.validation.Constraints;
import play.db.ebean.Model;
```

Com a anotação `Entity`, mapeamos a classe para uma tabela:

```
@Entity
public class Diretor extends Model {
```

E mapeamos também as colunas dessa tabela, sendo que a coluna `nome` é obrigatória.

```
private static final long serialVersionUID = 1L;
```

```
@Id
public Long id;
```

```
@Constraints.Required
public String nome;
```

Além disso, usando o recurso `Model.Finder` do Play, criaremos a variável auxiliar `find`, que auxiliará nas consultas por diretor:

```
public static Model.Finder<Long,Diretor> find =
    new Model.Finder<Long,Diretor>(Long.class,Diretor.class);
}
```

Por padrão, o Play implementa internamente um banco de dados em H2 para facilitar o desenvolvimento. Precisamos apenas descomentar do arquivo de configuração `application.conf` as seguintes linhas:

```
db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"
db.default.user=sa
db.default.password=""
ebean.default="models.*"
```

Logo após criarmos a classe `Diretor`, ao acessarmos aplicação, o Play sinaliza que precisamos atualizar (ou como ele chama, evoluir) o banco de dados para algo que represente as classes que foram criadas.

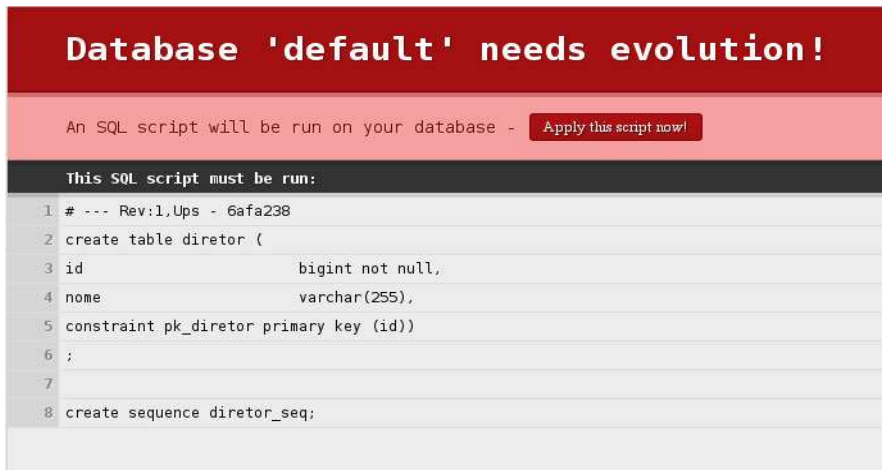


Figura 3.1: Evolution do play

Repare que a figura 3.1 mostra um script que cria uma tabela que representa fielmente a classe `Diretor`.

Clicando em `apply`, o seu banco de dados será atualizado em memória.

Rascunho de CRUD

Como estamos fazendo o Controller, criaremos apenas os métodos do nosso CRUD (*Create Retrieve Update Delete*) de diretores.

Inicialmente, a nossa classe tem apenas o método `lista`, que trará a lista de todos os diretores dos filmes.

```
package controllers;

import play.mvc.Controller;
import play.mvc.Result;

public class DiretorCRUD extends Controller {

    public static Result lista() {
        return TODO;
    }

}
```


Para chamar o método `lista`, vamos adicionar à página `inicio.scala.html` o trecho de código a seguir:

```
<ul class="nav navbar-nav">
  <li><a href="/diretor">Diretores</a></li>
  <li><a href="/sobre">Sobre o sistema</a></li>
```

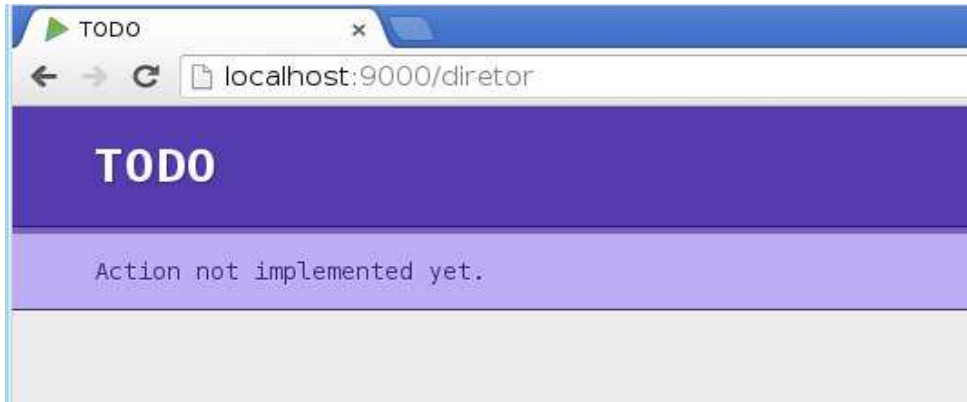


Figura 3.2: todo play

Acessando a página inicial do sistema, veremos o link `Diretores`. Ao clicar nele, veremos a figura 3.2, que mostra a tela do play de `TODO`, algo que é muito útil ao montar projetos! Com certeza essa tela é bem mais elegante do que levar um erro de `Página não encontrada` do servidor.

Listando

Uma característica interessante do Play é que os templates, assim como os métodos Java, podem ter parâmetros.

Precisamos enviar uma lista de diretores para o arquivo `diretor.scala.html` dentro de `app.views`, que fica logo no início:

```
@(diretores: List[Diretor])
```

Em seguida, chamamos o template `main` utilizado no 2, passando como parâmetro o título da página e o seu conteúdo HTML.

```
@main("Diretores") {
```

```
  <div class="container">
  <div class="header">
    <ul class="nav nav-pills pull-right">
```

Aqui criamos a rota para o cadastro de novo diretor:

```
  <li class="active">
    <a href="@routes.DiretorCRUD.novoDiretor()">
      Novo diretor
    </a>
  </li>
  <li class="active"><a href="/">Home</a></li>
</ul>
<h3 class="text-muted">Diretores</h3>
</div>
```

Nesse trecho definimos o loop que vai iterar entre cada diretor, exibindo o seu nome e um link para alterar cada um deles:

```
<table class="table table-striped table-bordered" id="example"
  cellpadding="0" cellspacing="0" border="0" width="100%">
<tfoot>
  @for(diretor <- diretores) {
    <tr>
      <th>G
      <a href="@routes.DiretorCRUD.detalhar(diretor.id)">
        @diretor.nome
      </a>
    </th>
  </tr>
  }
</tfoot>
</table>
</div>
}
```

O último passo é configurar a rota do link da página para o Controller, no arquivo de rotas `conf/routes`:

```
GET      /diretor                                controllers.DiretorCRUD.lista()
```

Vamos atualizar o método de listar os diretores no nosso controller:

```
public static Result lista() {
    List<Diretor> diretores = Diretor.find.findList();
    return ok(views.html.diretor.render(diretores));
}
```

A classe `Diretor` estende `play.db.ebean.Model`, que por sua vez, herda vários métodos que fazem o trabalho com o banco de dados, além de implementar a variável auxiliar `find`. Usando o método `findList`, temos a lista de todos os diretores armazenados.

Com isso, temos a nossa tela funcionando, mas como a tabela foi recém-criada, está vazia e não temos nada para ser exibido.

Vamos aprender em seguida como configurar um banco de dados.

Conectando

Para cada restart do servidor do Play, o seu banco de dados será limpo novamente. No início do desenvolvimento de um cadastro, isso não é um problema, mas chega um momento em que o sistema necessita de um cadastro mínimo já no banco de dados para funcionar de maneira adequada. Criar um cadastro de filmes em que é preciso cadastrar os diretores toda vez é inviável. Por esse motivo precisamos ter essas informações pré-cadastradas em um repositório.

Vamos agora configurar o nosso banco de dados para conectar-se ao PostgreSQL.

Para instalar o PostgreSQL e as tabelas do nosso exemplo, consulte o apêndice 11.

Novamente no arquivo `application.conf`, comentamos as linhas do H2 e adicionamos duas linhas do PostgreSQL:

```
#db.default.driver=org.h2.Driver
#db.default.url="jdbc:h2:mem:play"
#db.default.user=sa
#db.default.password=""
db.default.driver=org.postgresql.Driver
db.default.url="postgres://postgres:postgres@localhost/filmes"
```

Vamos também desativar o evolution, pois sempre no início da aplicação, o Play verifica se para cada classe model existe uma tabela em nosso banco de dados, e o nosso PostgreSQL já está corretamente configurado.

```
# Evolutions
# ~~~~~
# You can disable evolutions if needed
evolutionplugin=disabled
```

PLAY EVOLUTION

Note que o Play cria automaticamente um script SQL para cada mudança que for feita no seu modelo. Faça um refresh do seu projeto no workspace do Eclipse com `F5` e procure pelo arquivo `1.sql` dentro do diretório `conf-evolutions-default`.

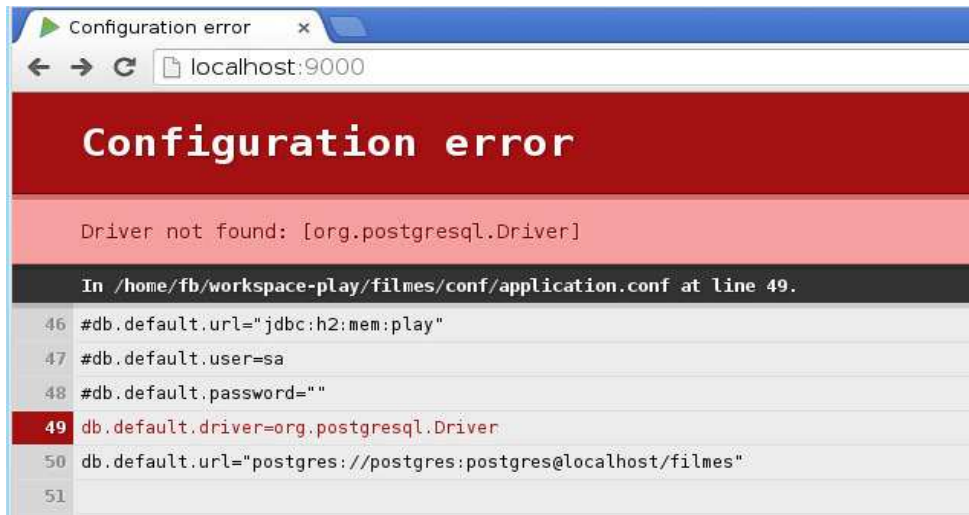


Figura 3.3: Play database driver

Acessando novamente a aplicação, deparamo-nos com o erro da figura 3.3, que mostra que existe uma dependência da nossa aplicação com o driver JDBC do PostgreSQL.

Para resolver esse problema de dependência, edite o arquivo `build.sbt` adicionando a linha do `postgresql` conforme o exemplo:

```
name := "filmes"
```

```

version := "1.0-SNAPSHOT"

libraryDependencies += Seq(
  javaJdbc,
  javaEbean,
  cache,
  "postgresql" % "postgresql" % "9.1-901-1.jdbc4"
)

play.Project.playJavaSettings

```

Ao fazer o restart do servidor, note no log que o download do driver foi feito (e guardado dentro do diretório do Play):

```

fb@cascao ~/workspace-play/filmes > play run
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
...
[info] downloading http://repo1.maven.org/maven2/postgresql/postgresql
/9.1-901-1.jdbc4/postgresql-9.1-901-1.jdbc4.jar ...
[info] [SUCCESSFUL]
    postgresql#postgresql;9.1-901-1.jdbc4!postgresql.jar
(1683ms)
...

```

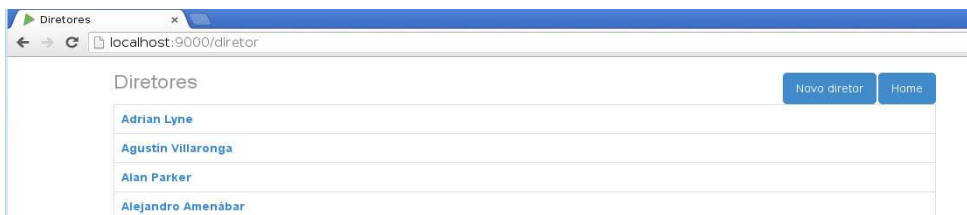


Figura 3.4: Lista de diretores

E a página de diretores finalmente é carregada do banco de dados conforme a figura 3.4.

Cadastrando

Vamos adicionar o cadastro de diretor. Para isso, vamos seguir os passos para construir uma tela no Play:

- 1) Criar um método no Controller;
- 2) Criar uma página template dentro do pacote views;
- 3) Configurar no arquivo de rotas o mapeamento da URI para o controller.

Para tratar com telas de cadastro e seus formulários, o Play tem um objeto chamado `Form` que encapsula parte da complexidade de trabalhar com formulários em HTML.

Na parte do controller, vamos adicionar um objeto do tipo `Form` para diretor:

```
private static final Form<Diretor> diretorForm =  
    Form.form(Diretor.class);
```

Em seguida, vamos adicionar dois métodos, o primeiro que apenas chama a tela de cadastro:

```
public static Result novoDiretor() {  
  
    return ok(views.html.novoDiretor.render(diretorForm));  
  
}
```

Já o o segundo faz a gravação no banco de dados. Vamos ver por partes. Inicialmente a variável `form` retornará as informações do diretor:

```
public static Result gravar() {  
  
    Form<Diretor> form = diretorForm.bindFromRequest();
```

Em seguida, é chamado o método que verifica se o `Form` veio com algum erro de validação, por exemplo, um campo obrigatório não preenchido. Caso ocorra algum erro de preenchimento, o Play automaticamente avisa com uma mensagem na tela.

Para exatamente essa necessidade, o Play tem um escopo chamado *flash*, no qual as informações nele colocadas estão disponíveis apenas em uma requisição. Com isso, adicionamos as mensagens de `sucesso` ou `erro`, que serão exibidas na tela conforme a necessidade.

Para a sua exibição correta, dentro do template `main.scala.html` temos as chamadas:

```

@if(flash.containsKey("sucesso")) {
  <div class="alert alert-success">
    @flash.get("sucesso")
  </div>
}

@if(flash.containsKey("erro")) {
  <div class="alert alert-danger">
    @flash.get("erro")
  </div>
}

```

Em seguida, se o formulário possuir erros, enviamos a mensagem adequada.

```

if (form.hasErrors()) {
  flash("erro", "Foram identificados problemas no cadastro");
  return ok(views.html.novoDiretor.render(diretorForm));
}

```

Depois temos a rotina de gravar os dados no banco, lembrando que a classe `Diretor` herdou o método `save`, que faz todo o trabalho de persistência.

```

Diretor diretor = form.get();
diretor.save();
// equivalente a ==> form.get().save();

```

E finalmente temos um registro de uma mensagem que será exibida na página, junto com o redirecionamento para a rota adequada.

```

flash("sucesso", "Registro gravado com sucesso");

return redirect(routes.DiretorCRUD.lista());
}

```

No começo do arquivo de template, informamos que ele tem como parâmetro um objeto do tipo `Form`. Lembrando que os templates estão na linguagem Scala, a sintaxe é um pouco diferente: `Form[Diretor]` em Scala é igual a `Form<Diretor>` em Java.

```

@(diretorForm : Form[Diretor])

```

```
@main("Novo diretor") {

    <div class="container">
        <div class="header">
            <ul class="nav nav-pills pull-right">
                <li class="active"><a href="/">Home</a></li>
            </ul>
        </div>
        <div>
```

Aqui usamos o helper para fazer a chamada do formulário ao nosso método de gravar dentro do controller.

```
@helper.form(action=routes.DiretorCRUD.gravar()) {
<form class="form-horizontal">
    <fieldset>

    <legend>Novo diretor</legend>
```

Além disso, ele é usado para informar o tipo de entrada de dados para o nome do diretor.

```
<div class="control-group">
    <div class="controls">
        @helper.inputText(diretorForm("nome"))
        <p class="help-block">informe o nome do diretor</p>
    </div>
</div>
}

<div class="control-group">
    <label class="control-label" for="singlebutton"></label>
    <div class="controls">
        <button id="singlebutton" name="singlebutton"
            class="btn btn-primary">
            Gravar
        </button>
    </div>
</div>

</div>
}
```


Finalmente vamos adicionar a rota desse cadastro. Note que para gravar estamos usando `POST`:

```
GET    /diretor/novo    controllers.DiretorCRUD.novoDiretor()
POST   /diretor/        controllers.DiretorCRUD.gravar()
```



The screenshot shows a web browser window with the title 'Novo diretor'. The address bar displays 'localhost:9000/diretor/novo'. The page content includes a heading 'Novo diretor', a label 'nome' above a text input field containing 'James Wan', a 'Required' label below the input, a placeholder text 'Informe o nome do diretor', and a blue button labeled 'Gravar'.

Figura 3.5: Novo cadastro de diretor

Ao clicar no link de novo diretor, veremos uma tela como na figura 3.5.

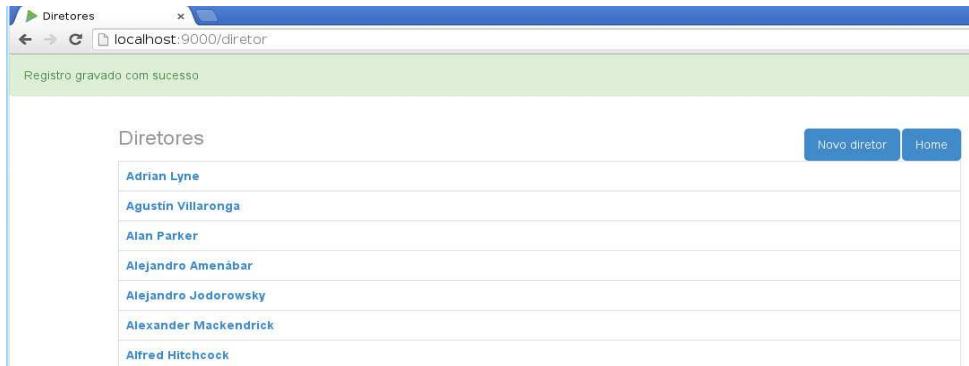


Figura 3.6: Cadastro de diretor feito com sucesso

Depois do cadastro, é exibida a mensagem de sucesso, como na figura 3.6.

Detalhando, alterando e removendo

O método do controller de detalhar é mais simples, é uma busca seguida de um redirecionamento:

```
public static Result detalhar(Long id) {
    Form<Diretor> dirForm =
    form(Diretor.class).fill(Diretor.find.byId(id));
    return ok(views.html.alterarDiretor.render(id,dirForm));
}
```

O método do controller de alterar é semelhante ao de cadastrar – a diferença é que chamamos o método `update` ao invés de `save`:

```
public static Result alterar(Long id) {
    form(Diretor.class).fill(Diretor.find.byId(id));

    Form<Diretor> alterarForm = form(Diretor.class).bindFromRequest();
    if (alterarForm.hasErrors()) {
        return badRequest(
            views.html.alterarDiretor.render(id,alterarForm));
    }
    alterarForm.get().update(id);
    flash("sucesso", "Diretor "
+ alterarForm.get().nome + " alterado com sucesso");
}
```

```

    return redirect(routes.DiretorCRUD.lista());
}

```

E o de remover apenas chamamos o método `delete`:

```

public static Result remover(Long id) {
    Diretor.find.ref(id).delete();
    flash("sucesso", "Diretor removido com sucesso");
    return lista();
}

```

O controller de alteração receberá os parâmetros de `id` e `Form<Diretor>`. Note que ele utiliza uma opção diferente do helper, com `import` (também aqui temos uma pequena diferença na sintaxe; em Scala `import helper._` é equivalente a `import helper.*` em Java).

```
@(id: Long, diretorForm: Form[Diretor])
```

```
@import helper._
```

```

@main("Alterar diretor") {

    <div class="container">
        <div class="header">
            <ul class="nav nav-pills pull-right">
                <li class="active"><a href="/">Home</a></li>
            </ul>
        </div>

        <h1>Alterar diretor</h1>

```

Aqui definimos a chamada ao método `alterar` do controller, e também voltamos para a listagem se for escolhida a opção cancelar:

```

@form(routes.DiretorCRUD.alterar(id)) {

    <fieldset>
        <inputText(diretorForm("nome"), '_label -> "Nome do diretor")
    </fieldset>

```

```
<div class="control-group">
  <div class="actions">
    <input type="submit" value="Gravar" class="btn btn-primary">
    <a href="@routes.DiretorCRUD.lista()" class="btn btn-primary">
      Cancelar
    </a>
  </div>
}
```

Aqui definimos a opção de remover o diretor, chamando o respectivo método do controller.

```
@form(routes.DiretorCRUD.remover(id)) {
  <input type="submit" value="Remover esse diretor"
    class="btn danger">
}

</div>
</div>
}
```

A última parte é definir as rotas restantes, portanto todas as rotas de diretor são:

GET	/diretor	controllers.DiretorCRUD.lista()
GET	/diretor/novo	controllers.DiretorCRUD.novoDiretor()
GET	/diretor/:id	controllers.DiretorCRUD.detalhar(id: Long)
POST	/diretor/:id	controllers.DiretorCRUD.alterar(id:Long)
POST	/diretor/	controllers.DiretorCRUD.gravar()
POST	/diretor/:id/remover	controllers.DiretorCRUD.remover(id:Long)

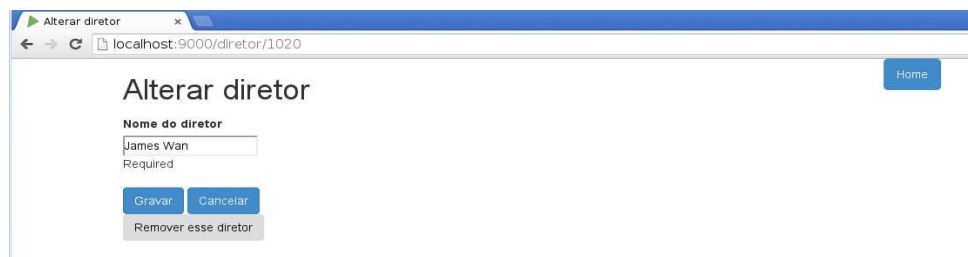


Figura 3.7: Cadastro de diretor feito com sucesso

A tela de alteração ficará como na figura 3.7, e exibirá mensagens após alterar ou remover algum registro de diretor.

Perceba que foram criados vários templates desse CRUD para ficar bem claro o papel de cada operação. Mas se necessário, é possível diminuir o número deles unificando algumas operações, como por exemplo a operação de novo diretor e alteração de diretor no mesmo template.

3.1 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a visão geral do Ebean (API de persistência) do Play Framework;
- como usar o objeto *Form* do Play;
- como trabalhar com templates e formulários do Play;
- como adicionar rotas com GET e POST.

Agora que já temos um produto razoavelmente viável criado, vamos publicar nas nuvens para ficar acessível em qualquer lugar do mundo!

CAPÍTULO 4

Publicando em qualquer lugar

Para publicar nossa aplicação (ou fazer deploy) no Play temos alguns cenários distintos.

O Play utiliza um servidor baseado no Netty, que é leve e robusto, suportando HTTP e HTTPS.

4.1 REDE LOCAL

Da mesma maneira que foi mostrado nos capítulos anteriores, um simples `play run` é suficiente para publicar sua aplicação rapidamente na rede local.

Se desejar rodar em outra porta HTTP, é preciso usar um parâmetro específico `play -Dhttp.port=<número-da-porta> run`:

```
fb@cascao ~/workspace-play/filmes > play -Dhttp.port=9999 run
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes (in build
```

```
file:/home/fb/workspace-play/filmes/)
```

```
--- (Running the application from SBT, auto-reloading is enabled) ---
```

```
[info] play - Listening for HTTP on /0.0.0.0:9999
```

```
(Server started, use Ctrl+D to stop and go back to the console...)
```

No caso da saída anterior, foi usado o `-Dhttp.port=9999`.

4.2 SERVIDOR JAVA EE

Até a versão do Play 2.1.4 era possível oficialmente gerar um pacote WAR que rodasse em um servidor como Apache Tomcat ou JBoss, mas nas versões posteriores isso foi descontinuado.

Existe o plugin não oficial `play2-war-plugin` que faz a conversão, mas não converte 100%. Ele não suporta, entre outras coisas, WebSockets.

Um WebSocket é uma implementação de mensagem assíncrona, bidirecional (full-duplex) em cima de uma conexão TCP, que foi especificada no Java pela JSR 356.

Se usar WebSocket for essencial, então oficialmente a única maneira de rodar é através do servidor do Play.

4.3 DEPLOY NA NUVEM

Não seria um absurdo se todo mês fosse cobrada uma taxa fixa de trezentos reais de energia elétrica, independente do uso ? Entretanto, quando se paga por um serviço de hospedagem, é exatamente o que acontece, pois o site fica no ar, independente do uso dos recursos por uma quantia mensal fixa. Para mudar esse cenário, surgiu o conceito de cloud computing (computação em nuvem), segundo o qual os serviços são pagos pelo uso, proporcionando inúmeras vantagens: toda a parte de infraestrutura (rede, armazenamento de arquivos, banco de dados, segurança) é de responsabilidade da prestadora de serviços, e o crescimento do uso de serviços é pago proporcionalmente, o que, em termos de custo, é uma excelente opção.

Para colocar o Play na nuvem, existem várias opções oficialmente documentadas, como Cloudbees, Cloud Foundry e Clever Cloud.

Vamos focar o nosso exemplo no *Heroku*.

Primeiros passos no Heroku

Acesse o site <https://www.heroku.com/> e crie uma conta gratuita.

Depois disso, acesse <https://toolbelt.heroku.com/> e baixe o conjunto de ferramentas para o seu sistema operacional.

Tudo no Heroku se faz com esse conjunto de ferramentas, e algumas coisas também é possível fazer pelo site.

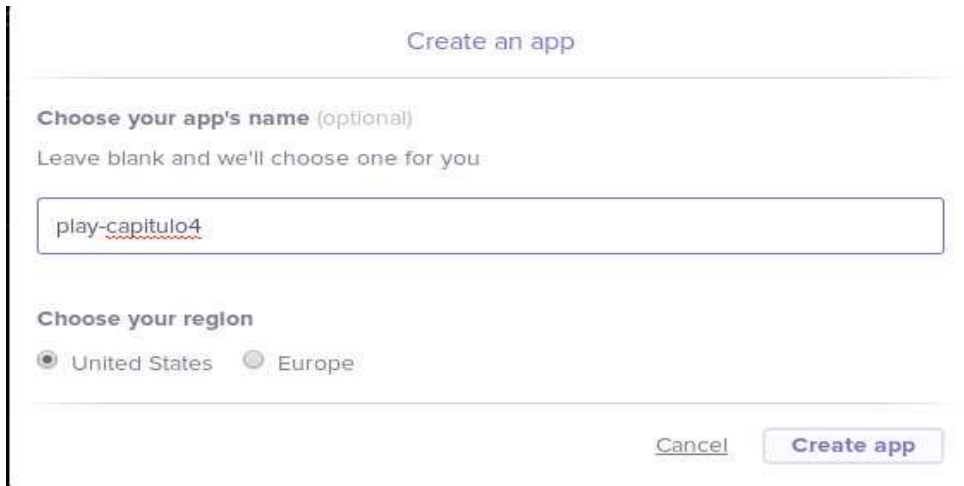
A screenshot of the Heroku 'Create an app' form. At the top, there is a link 'Create an app'. Below it, the section 'Choose your app's name (optional)' has a subtext 'Leave blank and we'll choose one for you'. A text input field contains 'play-capitulo4'. Below this, the section 'Choose your region' has two radio buttons: 'United States' (selected) and 'Europe'. At the bottom right, there are two buttons: 'Cancel' and 'Create app'.

Figura 4.1: Criando uma aplicação no Heroku

Pelo site é possível criar uma aplicação web. Vamos criar uma aplicação chamada “play-capitulo4” conforme a figura 4.1. Nesse exemplo foi criada a aplicação *play-capitulo4*. Crie um nome parecido, como por exemplo *<seu-usuario-no-Heroku>-play*.

Banco de dados

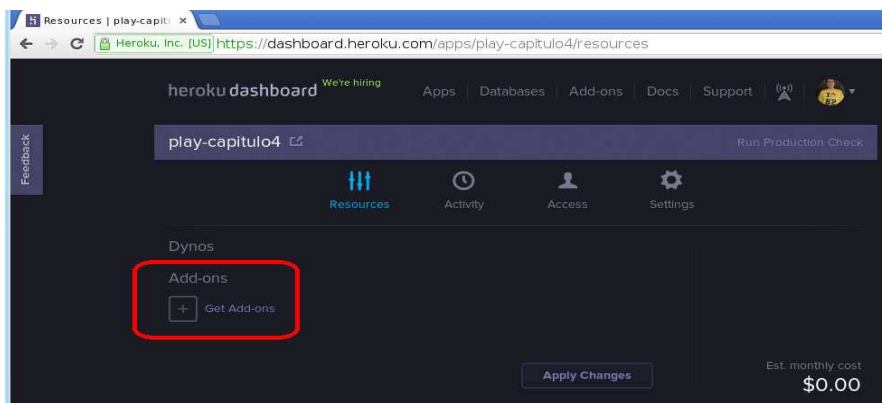


Figura 4.2: Adicionando funcionalidades à sua aplicação

Em seguida, adicionaremos uma melhoria ao site clicando na opção *Get add-ons* conforme a figura 4.2.

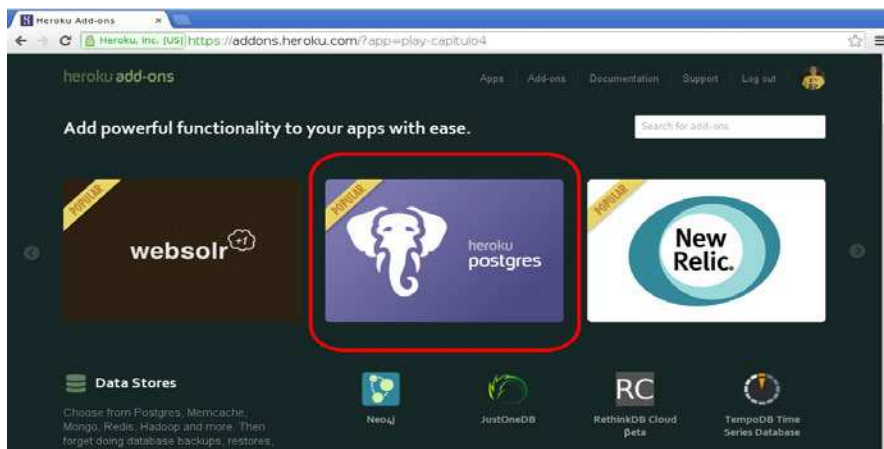


Figura 4.3: Escolhendo PostgreSQL para sua aplicação

Na lista de opções, escolha o banco de dados PostgreSQL conforme a figura 4.3.

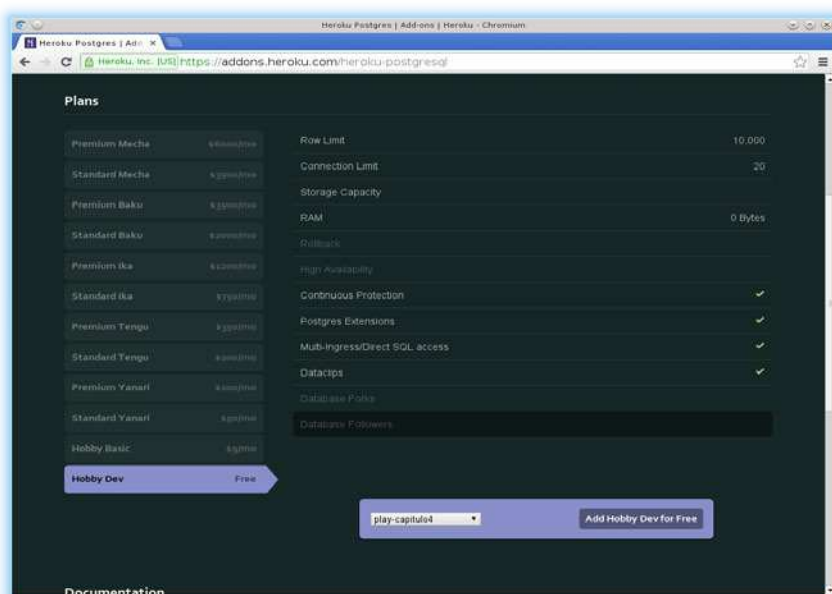


Figura 4.4: Adicionando funcionalidades à sua aplicação

Escolhendo a opção *Hobby Dev* e associada à aplicação recém-criada, teremos uma aplicação gratuita com o banco de dados PostgreSQL disponível com suporte a 20 conexões simultâneas e tabelas com no máximo 10.000 linhas (figura 4.4). O Heroku trabalha preferencialmente com PostgreSQL, mas oferece também outras opções como MySQL, MongoDB, Neo4J; é preciso consultar no site os planos existentes, mas a maioria deles oferece um gratuito de teste.

É possível escolher um plano com banco de dados de maior capacidade, mas nesse caso será cobrada uma taxa proporcional ao tamanho escolhido. Existem outras variações também: o MySQL, por exemplo, cobra dez dólares por um banco de um gigabyte; já o PostgreSQL, cobra nove dólares por um banco com dez milhões de linhas, independente de seu tamanho.

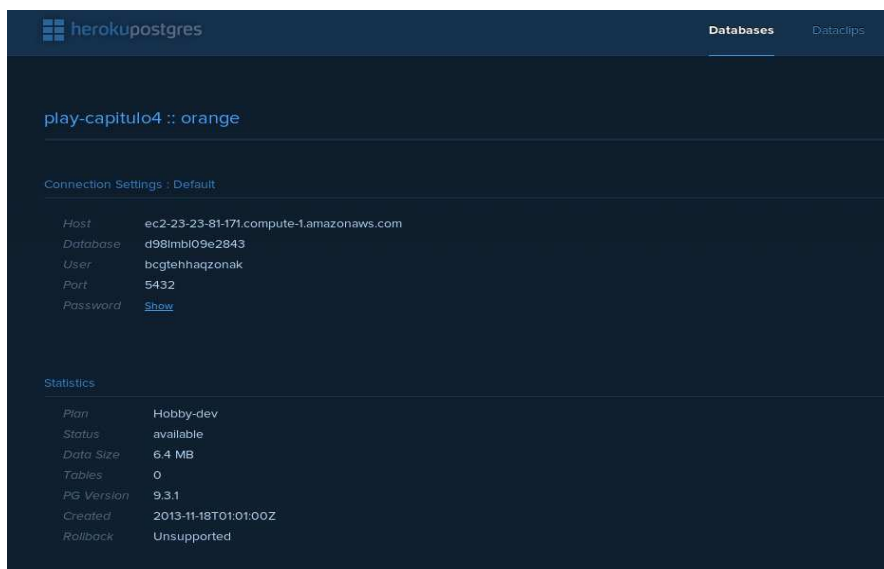


Figura 4.5: Adicionando funcionalidades à sua aplicação

Depois de criado, o banco de dados aparecerá na lista de *add-ons* da sua aplicação. Para obter informações de acesso ao banco de dados, clique no link *Heroku Postgres Hobby Dev* para exibir uma tela semelhante à figura 4.5.

Carga de dados

O banco de dados foi criado com sucesso, mas está vazio e precisa das tabelas de filmes e diretores para o correto funcionamento da aplicação.

Depois de baixarmos o script de <http://bit.ly/JLFA9C>, chamaremos o programa `psql` para fazer a carga de dados.

O `psql` é um programa que faz parte do PostgreSQL e é o interpretador de comandos SQL em modo texto, ou seja, é ele que envia os comandos SQL para o banco de dados. No apêndice 11 mostramos o PgAdmin3, que é o interpretador de comandos SQL em modo gráfico.

A sintaxe simplificada do `psql` é:

```
<diretório-de-instalação-postgresql>/psql
-h <servidor>
-U <usuário>
```

```
-d <banco-de-dados>
-p <porta> < <arquivo-SQL>
```

Exemplo acessando um banco de dados local chamado *filmes*:

```
C:\Program Files\PostgreSQL\9.3\bin>psql -U postgres -h
localhost -d filmes -p 5432
psql (9.3.1)
WARNING: Console code page (437) differs from Windows code page (1252)
        8-bit characters might not work correctly. See psql reference
        page "Notes for Windows users" for details.
Type "help" for help.

filmes=#
```

Exemplo acessando um banco de dados na nuvem e rodando um script de comandos SQL:

```
C:\Program Files\PostgreSQL\9.3\bin\psql
-h ec2-23-23-81-171.compute-1.amazonaws.com
-U bcgtehhaqzonak
-d d98lmb109e2843
-p 5432 < C:\SQL\banco-filmes-postgresql.sql
Senha para usuário bcgtehhaqzonak:
CREATE TABLE
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
...
```

Depois de executado o script, foram criadas duas tabelas em nosso banco de dados e populadas com alguns registros.

Acessando novamente a tela da figura 4.5, verificamos que a quantidade de tabelas aumentou de zero para dois, o que comprova a execução do script com sucesso.

Deploy da aplicação

Para o processo de deploy (publicação) da aplicação, precisamos criar um arquivo de configuração do Play, diferenciado para esse ambiente, e outro arquivo que orienta o Heroku como subir a aplicação.

No diretório de configurações (o mesmo que se encontra o `application.conf`), criaremos o arquivo `heroku.conf` com as informações de acesso ao banco de dados:

```
include "application.conf"
# db.default.url="postgres://usuario:senha@servidor/nomeDoBanco"
db.default.url=
  "postgres://bcgtezonak:senha@comp.amazonaws.com/d98lmb109e2843"
```

Depois disso, na raiz do projeto criaremos o arquivo `Procfile` contendo as informações

```
web: target/universal/stage/bin/filmes
-Dhttp.port=$PORT
-Dconfig.resource=heroku.conf
```

Para usar os serviços do Heroku, é preciso instalar o seu conjunto de ferramentas, disponível para download em <https://toolbelt.heroku.com/>, que contém os programas *heroku* e *git*.

No Heroku não existe o conceito de stop e start do servidor, isso é feito quando você commita alguma alteração no seu código.

Portanto, basicamente precisamos nos logar no Heroku conforme o exemplo:

```
fb@cascao ~/workspace-play/filmes-cap04 > heroku auth:login
Enter your Heroku credentials.
```

```
Authentication successful.
fb@cascao ~/workspace-play/filmes-cap04 >
```

Em seguida, adicione o seu projeto ao do Heroku pelo git:

```
git remote add heroku git@heroku.com:play-capitulo4.git
```

E para conferir se foi adicionado corretamente, o servidor deve aparecer conforme o exemplo:

```
fb@cascao ~/workspace-play/filmes-cap04 > git remote -v
heroku  git@heroku.com:play-capitulo4.git (fetch)
heroku  git@heroku.com:play-capitulo4.git (push)
```

SUPOORTE AO JAVA 7

Se desejar que no Heroku a sua aplicação rode com o Java 7, adicione na raiz do projeto um arquivo chamado *system.properties* contendo uma única linha: *java.runtime.version=1.7*.

Depois disso, precisamos apenas commitar as alterações e enviar para o Heroku pelo Git, o restante será feito automaticamente.

```
fb@cascao ~/workspace-play/filmes-cap04 > git commit -a -m "capitulo-04"
```

```
fb@cascao ~/workspace-play/filmes-cap04 > git push heroku master
```

```
Enter passphrase for key '/home/fb/.ssh/id_dsa':
warning: no threads support, ignoring --threads
Counting objects: 5, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)

-----> Play 2.x - Java app detected
-----> Installing OpenJDK 1.7...done
-----> Running: sbt clean compile stage
        Getting org.scala-sbt sbt 0.13.0 ...
...
[info] Done packaging.
[succes] Total time: 6 s, completed Nov 23, 2013 1:15:21 AM
-----> Dropping ivy cache from the slug
-----> Dropping project boot dir from the slug
-----> Dropping compilation artifacts from the slug
-----> Discovering process types
        Procfile declares types -> web

-----> Compiled slug size: 98.3MB
-----> Launching... done, v9
```

```
http://play-capitulo4.herokuapp.com deployed to Heroku
```

```
To git@heroku.com:play-capitulo4.git  
e43afcf..aea7172 master -> master
```

Em seguida, acessando o link <http://play-capitulo4.herokuapp.com/> (ou <http://<seu-usuario-no-Heroku>-play.herokuapp.com/>) veremos a nossa aplicação funcionando perfeitamente conforme a figura 4.6.



Figura 4.6: Aplicação na nuvem

4.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a mudar a porta HTTP do servidor do Play;
- a criar uma aplicação no Heroku;
- a criar e popular uma base de dados PostgreSQL no Heroku;
- a configurar uma aplicação Play para rodar no Heroku.

Agora que finalizamos o Quickstart, vamos aprender a melhorar nossa aplicação nos próximos capítulos.

CAPÍTULO 5

Melhorando o input do usuário

O nosso sistema ainda tem alguns problemas: não existe cadastro de filmes e se tentarmos remover algum diretor, ocorre o erro da figura 5.1.



The screenshot shows a red error banner at the top with the text "Execution exception". Below the banner, the error message is displayed in a light red box: "[PersistenceException: ERROR executing DML bindLog[] error[ERRO: atualização ou exclusão em tabela "diretor" viola restrição de chave estrangeira "diretor_fk" em "filmes_cult"\n Detalhe: Chave {id}=(2) ainda é referenciada pela tabela "filmes_cult".]]". Below the error message, the location of the exception is shown: "In ./home/fb/workspace-play/filmes-cap04/app/controllers/DiretorCRUD.java at line 23.". The code snippet is shown in a light gray box with line numbers 20 through 28. Line 23 is highlighted in red. The code is as follows:

```
20     }
21
22     public static Result remover(Long id) {
23         Diretor.find.ref(id).delete();
24         flash("sucesso", "Diretor removido com sucesso");
25         return lista();
26     }
27
28     public static Result novoDiretor() {
```

Figura 5.1: Play com erro de validação

Vamos criar o cadastro de filmes usando mais componentes, mas dessa vez cus-

tomizando os próprios helpers existentes do Play para o nosso proveito.

A classe Controller do nosso CRUD será bem semelhante à de Diretor, apenas na lista inicial vamos ordenar os filmes pelo ano, por isso chamaremos o método passando o parâmetro que desejamos ordenar:

```
public class FilmeCRUD extends Controller {

    public static Result lista() {
        List<Filme> filmes = Filme.find.where()
                                   .orderBy("ano").findList();
        return ok(views.html.filme.render(filmes));
    }
}
```

Note também o método `gravar`, que é bem semelhante ao de Diretor, apesar de gravar muito mais campos. Essa facilidade acontece logo na primeira linha, quando o método `bindFromRequest` relaciona os campos informados na tela com o objeto `form`.

```
public static Result gravar() {

    Form<Filme> form = filmeForm.bindFromRequest();
    if (form.hasErrors()) {

        flash("erro",
            "Foram identificados problemas no cadastro de filme");

        List<Diretor> diretores = Diretor.find.findList();

        return ok(views.html.novoFilme.render(filmeForm,diretores));
    }

    form.get().save();

    flash("sucesso","Registro gravado com sucesso");

    return redirect(routes.FilmeCRUD.lista());
}
```

Depois `bindFromRequest`, temos a validação dos campos com o `form.getErrors`. Se existir algum erro, é exibida a mensagem colocando no escopo `flash`.

Em seguida, uma pequena otimização de código: chamamos o método `save` direto do `form` para gravar o registro.

Novo filme



O formulário "Novo filme" contém os seguintes campos:

- Ano:** Seletor de lista suspensa com o valor "1987" e uma seta para baixo.
- Filme:** Campo de texto com o valor "Full Metal Jacket".
- Diretor:** Seletor de lista suspensa com o valor "Stanley Kubrick" e uma seta para baixo.
- Duração:** Campo de texto com o valor "116".
- Gênero:** Campo de texto com o valor "Drama".

Na base do formulário, há dois botões azuis: "Gravar" e "Cancelar".

Figura 5.2: Cadastro de um filme

Para exibirmos a tela da figura 5.2, usaremos o mesmo helper existente, porém vamos customizar criando um arquivo de template `campo.scala.htm`.

Aqui colocamos os `div` e estilos conforme o estilo do Bootstrap 3:

```
@(elements: helper.FieldElements)
<div class="@if(elements.hasErrors) {error} else {form-group}">
  <label for="@elements.id" class="col-sm-2 control-label">
    @elements.label
  </label>
  <div class="col-sm-10">
    @elements.input
  </div>
</div>
```

E no arquivo `novoFilme.scala.html` existem dois parâmetros, o formulário de filmes e a lista de diretores, que será utilizada em um combo.

```
@(filmeForm : Form[Filme],diretores: List[Diretor])
```

```
@implicitField = @{ FieldConstructor(campo.f) }
```

```
@import helper._
```

Esse trecho não apresenta novidades, apenas as chamadas do formulário e o link para a página inicial.

```
@main("Novo filme") {
```

```
<div class="container">
```

```
<ul class="nav nav-pills pull-right">
```

```
<li class="active"><a href="@routes.FilmeCRUD.lista()">Home</a></li>
```

```
</ul>
```

```
<div class="row clearfix">
```

```
<div class="col-md-12 column">
```

```
@form(action=routes.FilmeCRUD.gravar(), 'class->"form-horizontal") {
```

```
<fieldset>
```

```
<legend>Novo filme</legend>
```

Aqui usamos dois helpers para montar os componentes da tela: o `@inputText` para gerar os campos texto do nome, duração e gênero do filme, e `@select` para montar os combos de diretor e ano.

```
@select(
```

```
    filmeForm("ano"),
    options(Filme.anoOptions),
    '_label -> Messages("filme.ano"),
    '_default -> "-- Ano --"
```

```
)
```

```
@inputText(filmeForm("nome"), '_label -> "Filme")
```

```
@select(
```

```
    filmeForm("diretor.id"),
    options(Diretor.options),
```

```
        '_label -> "Diretor", '_default -> "-- Diretor --"
    )
```

```
@inputText(filmeForm("duracao"), '_label -> "Duração")
```

```
@inputText(filmeForm("genero"), '_label -> "Gênero")
```

Aqui finalizamos o arquivo sem novidades, com os botões de gravar e cancelar semelhantes ao da tela de diretor.

```
</fieldset>
```

```

<div class="form-group">
  <label class="col-md-3 col-md-3 control-label"
        for="singlebutton"> </label>
  <div class="col-lg-6">
    <button id="singlebutton" name="singlebutton"
            class="btn btn-primary">
      Gravar
    </button>
    <a href="@routes.FilmeCRUD.lista()" class="btn btn-primary">
      Cancelar
    </a>
  </div>
</div>

```

```

    }
  </div>
}

```

Na maioria das telas temos alguns textos em comum, como a palavra `Home` ou `Gravar`. Para alterar de “Gravar” para “Efetivar” no sistema, é necessário alterar manualmente em todas as páginas.

Para facilitar esse tipo de trabalho, o Play permite isolar todas as mensagens em um único arquivo, o que simplifica a manutenção do sistema.

5.1 ISOLANDO MENSAGENS

Para configurarmos o nosso sistema para isolar as mensagens, basta editarmos o arquivo `application.conf` para adicionarmos os idiomas suportados:

```
#application.langs="en"
application.langs="en,pt"
```

Em seguida, no mesmo diretório do arquivo de configuração anterior, criaremos um arquivo de mensagens com a extensão do idioma suportado.

Para a nossa aplicação, usaremos o arquivo `messages.pt`:

```
#
# Mensagens
#

# globais
global.home=Home
global.erro=Erro interno do sistema
global.gravar=Gravar
global.cancel=Cancelar

# filme
filme.duracao=Duração
filme.novo=Novo filme
filme.ano=Ano
filme.nome=Filme
filme.diretor=Diretor
filme.nome.hint=informe o nome do filme
filme.genero=Gênero
filme.ano.combo=-- Ano --
filme.diretor.combo=-- Diretor --
```

Com essa alteração, o arquivo `novoFilme.scala.html` ficaria assim:

```
@(filmeForm : Form[Filme],diretores: List[Diretor])

@implicitField = @{ FieldConstructor(campo.f) }

@import helper._

@main(Messages("filme.novo")) {

<div class="container">

  <ul class="nav nav-pills pull-right">
    <li class="active">
```

```
<a href="@routes.FilmeCRUD.lista()">
  Home
</a>
</li>
</ul>

<div class="row clearfix">
  <div class="col-md-12 column">

    @form(action=routes.FilmeCRUD.gravar(), 'class->"form-horizontal") {

      <fieldset>

        <legend>@Messages("filme.novo")</legend>

        @select(
          filmeForm("ano"),
          options(Filme.anoOptions),
          '_label -> Messages("filme.ano"),
          '_default -> Messages("filme.ano.combo")
        )

        @inputText(filmeForm("nome"),
          '_label -> Messages("filme.nome"))

        @select(
          filmeForm("diretor.id"),
          options(Diretor.options),
          '_label -> Messages("filme.diretor"), '_default ->
            Messages("filme.diretor.combo")
        )

        @inputText(filmeForm("duracao"),
          '_label -> Messages("filme.duracao"))

        @inputText(filmeForm("genero"),
          '_label -> Messages("filme.genero"))

      </fieldset>

      <div class="form-group">
```

```

<label class="col-md-3 col-md-3 control-label"
      for="singlebutton"> </label>
<div class="col-lg-6">
  <button id="singlebutton" name="singlebutton"
        class="btn btn-primary">
    @Messages("global.gravar")
  </button>
  <a href="@routes.FilmeCRUD.lista()"
    class="btn btn-primary">
    @Messages("global.cancel")
  </a>
</div>
</div>
}
</div>
}

```

Além de usarmos o `@Messages` para trocar os textos na tela, também trocamos os valores texto passados como parâmetro, como foi feito no início substituindo `@main("Novo filme")` por `@main(Messages("filme.nome"))`.

5.2 TRATANDO ERROS

Se tentarmos remover algum diretor, ocorre o erro da figura 5.1.

Para corrigirmos esse problema, basta tratarmos qualquer tipo de `Exception` e devolvermos para a tela uma mensagem padrão.

Na classe *DiretorCRUD*, alteramos a rotina de remover diretor adicionando o tratamento adequado:

```

public static Result remover(Long id) {
    try {
        Diretor.find.ref(id).delete();
        flash("sucesso","Diretor removido com sucesso");
    } catch (Exception e) {
        flash("erro",play.i18n.Messages.get("global.erro"));
    }
    return lista();
}

```

Com isso, obteremos o resultado esperado com uma mensagem amigável ao usuário, como mostra a figura 5.3.



Figura 5.3: Play com erro de validação tratado

5.3 PÁGINAS CUSTOMIZADAS

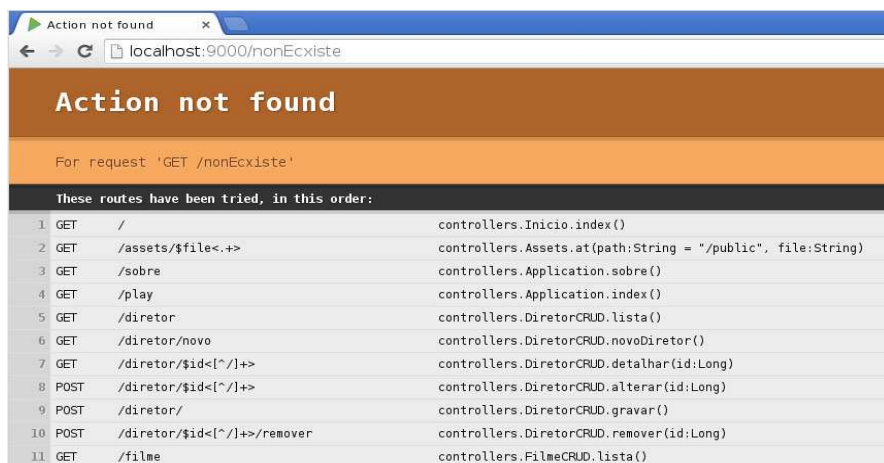


Figura 5.4: Página não encontrada padrão

Se tentarmos acessar uma URL que não existe, o Play automaticamente lista todas as actions existentes, como na figura 5.4.

Entretanto, não é interessante deixar isso como padrão, pois mostra (de maneira nada segura) as diferentes maneiras de acesso à aplicação, além de confundir o usuário e não exibir um link para retornar.

Felizmente, esse comportamento pode ser facilmente alterado sobrescrevendo com a classe `GlobalSettings`, que é responsável por definir opções globais ao sistema todo.

Para alterar essas configurações, basta criarmos uma classe chamada `Global` e estender `GlobalSettings`.

Se, por exemplo, desejarmos exibir um aviso quando a aplicação subir, basta sobrescrevermos o método:

```
import play.*;

public class Global extends GlobalSettings {

    @Override
    public void onStart(Application app) {

        System.out.println("Top 100 filmes cult no ar!");

    }

}
```

Entretanto, para sobrescrever os métodos, devemos obedecer à API do Play, utilizando a mesma assinatura dos métodos conforme a documentação em <http://www.playframework.com/documentation/2.2.1/api/java/play/GlobalSettings.html>.

Para sobrescrever o comportamento de página não encontrada, precisamos usar a assinatura com retorno do tipo `Promise<SimpleResult>` (`Promise` é uma classe interna em Scala do Play, e `Promise.pure` usado dentro do método também):

```
import static play.mvc.Results.notFound;
import play.GlobalSettings;
import play.libs.F.Promise;
import play.mvc.Http.RequestHeader;
import play.mvc.SimpleResult;

public class Global extends GlobalSettings {

    @Override
    public Promise<SimpleResult> onHandlerNotFound(
        RequestHeader request) {
        return Promise.<SimpleResult> pure(
            notFound(views.html.paginaNaoEncontrada.render(request.uri()))
        );
    }

}
```

Essa classe utiliza a seguinte página customizada, `paginaNaoEncontrada.scala.html`:

```
@(url: String)

<html>
<body>
  <h1>Oops, essa página não existe...</h1>

  <p>@url</p>

  <p> <a href="/">Top 100 filmes Cult</a></p>
</body>
</html>
```

E obtemos o resultado da figura 5.5:



Figura 5.5: Página não encontrada customizada

5.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a utilizar os helpers de input de formulários do Play;
- a isolar as mensagens da aplicação;

- a customizar página padrão do sistema.

Agora que já temos uma validação de entrada melhorada, vamos expor alguns serviços de nosso sistema e acessá-los externamente.

CAPÍTULO 6

Criando serviços

Quando acessamos um site com um celular, por mais moderno que ele seja, não é a mesma coisa que com o desktop, principalmente pelo tamanho da tela.

Com isso surge a necessidade de disponibilizarmos as informações do site de uma maneira mais amigável para que uma aplicação de celular use esses dados como bem entender.

Por esse motivo, vamos mostrar aqui como podemos facilmente disponibilizar as informações e como acessar de uma aplicação Android.

6.1 ACESSANDO SERVIÇO VIA WEB

Existem diversas maneiras de disponibilizar as informações do nosso site, vamos tratar dos dois formatos mais comuns: XML e JSON (*JavaScript Object Notation*).

Começaremos criando um controller chamado `Services` específico para esses dois serviços, ambos usam a mesma rotina de obter a lista de filmes ordenados pelo ano:

```
public class Services extends Controller {  
  
    private static List<Filme> getFilmesOrderByAno() {  
        return Filme.find.where().orderBy("ano").findList();  
    }  
  
}
```

Para exibirmos a lista de filmes em XML, formatamos o retorno manualmente:

```
public static Result listaFilmesEmXML() {  
  
    List<Filme> filmes = getFilmesOrderByAno();  
    return ok("<message \"status\"=\"OK\"> " +  
        filmes.toString() + "</message>");  
  
}
```

Para exibir o resultado em JSON, usamos a rotina auxiliar `Json.toJson` para formatar a lista de filmes.

```
public static Result listaFilmesEmJSON() {  
  
    List<Filme> filmes = getFilmesOrderByAno();  
    return ok(Json.toJson(filmes));  
  
}
```

Acessando a URL `http://<ip-da-sua-maquina>:9000/filmesJSON` via browser, vemos as informações do filme como mostra a figura 6.1.

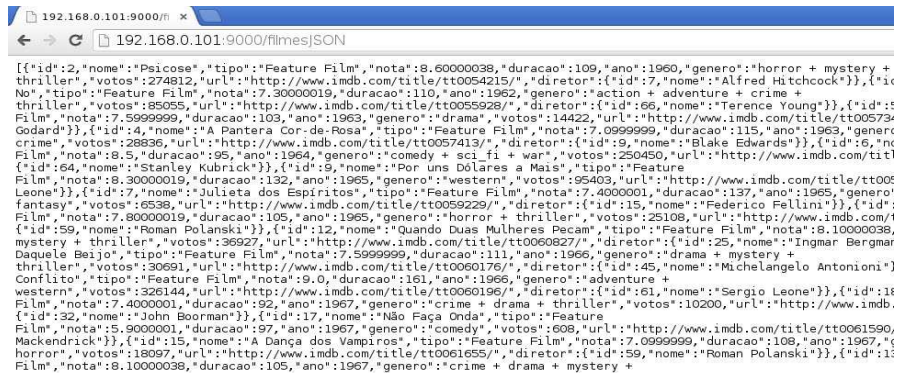


Figura 6.1: Lista de filmes em formato JSON

Entretanto, estamos aqui dividindo o sistema em duas partes, a que disponibiliza serviços (a web) e a que consome os serviços (a mobile – Android), e precisamos verificar se o nosso serviço está funcionando corretamente.

Uma maneira eficaz de fazer essa verificação é depurar (ou debugar) a aplicação e conferir se os dados enviados estão corretos.

Portanto, vamos rapidamente configurar o Eclipse para depurar a nossa aplicação.

6.2 DEBUCANDO PELO ECLIPSE

Já sabemos que para subir a aplicação usamos no console o comando `play run`:

```
fb@cascao ~/workspace-play/filmes > play run
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
```

```
--- (Running the application from SBT, auto-reloading is enabled) ---
```

```
[info] play - Listening for HTTP on /0.0.0.0:9000
```

(Server started, use Ctrl+D to stop and go back to the console...)

Para subir em modo debug, usamos no console o comando `play debug run`:

```
fb@cascao ~/workspace-play/filmes > play debug run
Listening for transport dt_socket at address: 9999
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
```

--- (Running the application from SBT, auto-reloading is enabled) ---

```
[info] play - Listening for HTTP on /0.0.0.0:9000
```

(Server started, use Ctrl+D to stop and go back to the console...)

Agora, além de a aplicação estar disponível na porta 9000, a porta de debug 9999 está liberada para acessar.

Pelo Eclipse, na opção *Debug Configurations*, vamos configurar uma conexão remota conforme a figura 6.2.

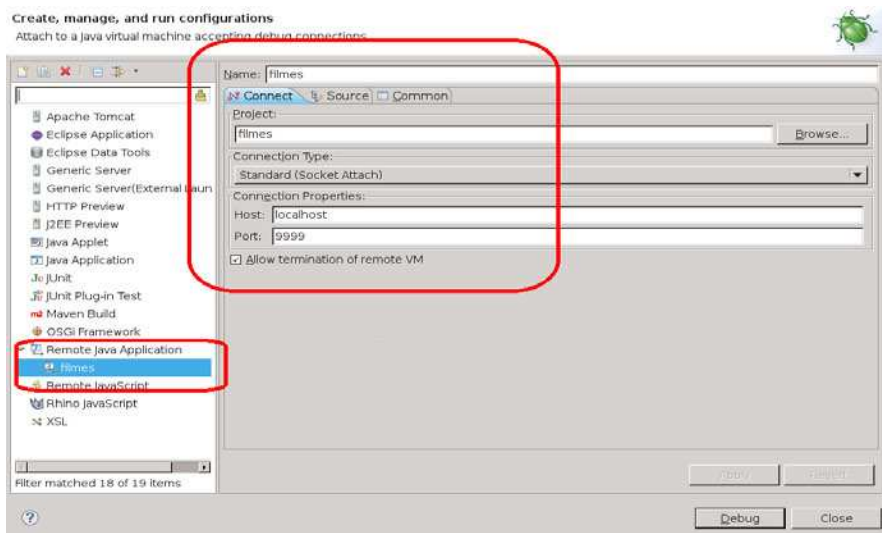


Figura 6.2: Configurando a aplicação em debug

Colocando um breakpoint nos serviços, visualizamos as informações conforme a figura 6.3.

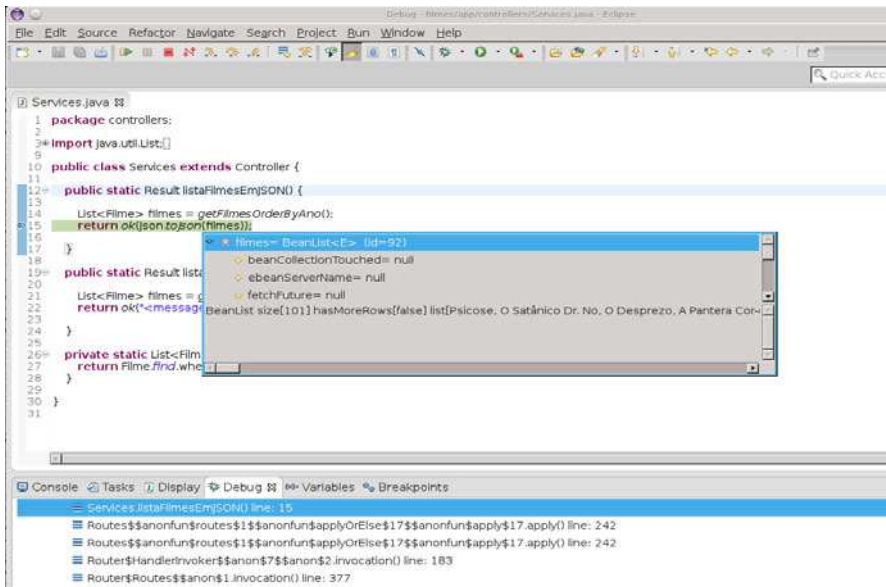


Figura 6.3: Aplicação em debug

6.3 ACESSANDO SERVIÇO VIA MOBILE

Vamos utilizar uma simples aplicação de celular para usar um serviço nosso.

Para montar o seu ambiente Android e chamar o emulador da aplicação mobile, consulte o apêndice 12.

Inicialmente, carregamos os dados do serviço e os atribuímos a uma variável do tipo `JSONArray`. Isso é feito chamando o método `JsonParser.getJSONFromUrl` e passando como parâmetro a URL `http://<ip-da-sua-maquina>:9000/filmesJSON` que devolverá o conteúdo das informações no formato JSON.

```

public class Top100FilmesCult extends ListActivity
    implements OnClickListener {

    @Override
    public void onClick(View arg0) {

        try {

```



```
JSONParser jParser = new JSONParser();

JSONArray json = jParser.getJSONFromUrl(
    "http://192.168.0.101:9000/filmesJSON");
```

Depois, cada filme é lido do formato JSON e adicionado à lista na tela dinamicamente:

```
for (int i = 0 ; i < json.length() ; i++) {

    try {

        JSONObject c = json.getJSONObject(i);
        String nome = c.getString(NOME);
        String ano = c.getString(ANO);

        listItems.add(ano + " - " + nome);

    } catch (JSONException e) {
        e.printStackTrace();
    }

}

} catch (Exception e) {
    e.printStackTrace();
}

adapter.notifyDataSetChanged();

}
```

E temos como resultado a figura 12.4 exibindo a lista de filmes.



Figura 6.4: Lista de filmes no Android

6.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a expor um serviço em formato XML e JSON;
- a debugar uma aplicação Play no Eclipse.

Agora que já temos serviços expostos, vamos integrá-los às redes sociais para melhor difusão de nosso sistema!

CAPÍTULO 7

Integrando nas redes sociais

As redes sociais são essenciais para a divulgação do site e seus serviços. Vamos integrar o nosso site ao Facebook de duas maneiras, primeiramente utilizando a API nativa disponibilizada para os desenvolvedores, e depois utilizando um plugin do Play para conectar a várias redes sociais: Twitter, GitHub, Google, LinkedIn, Foursquare, Instagram, VK (rede social russa semelhante ao Facebook) e XING (rede social semelhante ao LinkedIn).

7.1 CRIANDO UMA APLICAÇÃO NO FACEBOOK

Qualquer integração ao Facebook é feita cadastrando-se uma aplicação, o que pode ser facilmente feito no link <https://developers.facebook.com/apps/>.

Devemos preencher as informações da aplicação, como mostra a figura 7.1. As informações são simples: o primeiro campo é o nome do aplicativo, o segundo é o namespace (se desejar ter uma URL <http://apps.facebook.com/namespace>), e finalmente, o terceiro é para classificar sua categoria. No exemplo usamos a categoria

de `Diversão`, mas se fosse um aplicativo de tênis, para exemplificar, a categoria adequada seria `Esporte`.

Figura 7.1: Cadastro de aplicação no Facebook

No capítulo 4 publicamos uma aplicação básica (<http://play-capitulo4.herokuapp.com/>), e usaremos agora uma URL diferente para refletir a aplicação completa (<http://top100filmescult.herokuapp.com/>). Depois de cadastrados os dados iniciais, devemos informar um e-mail de contato e o domínio no qual a aplicação funcionará. Será necessário cadastrar um domínio diferente, por exemplo `<seu-usuario-no-Heroku>-top100filmescult.herokuapp.com`, semelhante à figura 7.2.



The screenshot shows the Facebook App registration interface. At the top, the app name 'top100filmscult' is displayed next to its icon. Below this, the 'App ID' is 260229664128047, the 'App Secret' is 08b7b1d37ff5ca9bc95b21648a3 (with a 'redefinir' link), and a status message indicates the app is in 'Sandbox Mode' (only visible to Admins, Developers and Testers). A section titled 'Informações básicas' contains several input fields: 'Display Name' (top100filmscult), 'Namespace' (topfilmscult), 'Contact Email' (boaglio@gmail.com), 'App Domains' (top100filmscult.herokuapp.com), and 'Sandbox Mode' (set to 'Ativada'). Below this is a section 'Select how your app integrates with Facebook' with a checked option for 'Website with Facebook Login' and a 'Site URL' field containing http://top100filmscult.herokuapp.com.

Figura 7.2: Cadastro de aplicação no Facebook com domínio

Após o cadastro com sucesso, nenhuma mensagem aparece na tela, apenas a tela é recarregada e o aplicativo recém-criado aparece na lista.

7.2 INTEGRAÇÃO VIA JAVASCRIPT

A integração via JavaScript é algo que pode ser feito independente de linguagem ou framework utilizado, ou seja, se a nossa aplicação fosse em ASP.Net ou PHP, seria indiferente.

Vamos colocar na lista de filmes do site uma parte de comentários do Facebook. O usuário usa a tela como uma coisa só, mas na verdade vamos inserir um trecho de código JavaScript (que se encarregará de gravar e exibir os comentários), editando o arquivo `filme.scala.html`. Inicialmente adicionamos a chamada rotina informando no campo `appId` o valor da chave que está na figura 7.1.

O código está disponível no Quickstart: Facebook SDK for JavaScript no link <https://developers.facebook.com/docs/javascript/quickstart>. Vamos adaptá-lo ao nosso exemplo.

```
<script>
```

```

window.fbAsyncInit = function() {
  // init the FB JS SDK
  FB.init({
    appId      : '260229664128047', // App ID from the app dashboard
    status     : true,              // Check Facebook Login status
    xfbml      : true               // Look for social plugins on ...
  });

  // Additional initialization code such as adding
  // Event Listeners goes here
};

```

Em seguida, vamos chamar o restante da rotina do Facebook:

```

// Load the SDK asynchronously
(function(){
  // If we've already installed the SDK, we're done
  if (document.getElementById('facebook-jssdk')) {return;}

  // Get the first script element, which we'll use
  // to find the parent node
  var firstScriptElement =
    document.getElementsByTagName('script')[0];

  // Create a new script element and set its id
  var facebookJS = document.createElement('script');
  facebookJS.id = 'facebook-jssdk';

  // Set the new script's source to the source of the Facebook JS SDK
  facebookJS.src = '//connect.facebook.net/en_US/all.js';

  // Insert the Facebook JS SDK into the DOM
  firstScriptElement.parentNode.insertBefore(facebookJS,
                                             firstScriptElement);

})();
</script>

<div id="fb-root"></div>
<script>(function(d, s, id) {
  var js, fjs = d.getElementsByTagName(s)[0];
  if (d.getElementById(id)) return;
  js = d.createElement(s); js.id = id;

```

```
js.src =  
  "//connect.facebook.net/pt_BR/all.js#xfbml=1&appId=260229664128047";  
fjs.parentNode.insertBefore(js, fjs);  
(document, 'script', 'facebook-jssdk'));
```

E no final da página adicionaremos o `div fb-comments` que exibirá o campo de comentários.

```
<center>  
  <div  
    class="fb-comments"  
    data-href="http://top100filmescult.herokuapp.com/filme"  
    data-numposts="5"  
    data-colorscheme="light">  
  </div>  
</center>  
</div>
```

E temos como resultado a opção de comentários, conforme é ilustrado na figura 73.

The screenshot shows a web browser window with the address bar displaying 'top100filmscult.herokuapp.com/filme'. Below the browser window is a table of movies with columns for year, title, director, rating, and genres. Below the table is a Facebook social plugin for comments.

1994	Riget		280	comedy + drama + fantasy + horror + mystery
1994	Nightwatch - Perigo na Noite	Ole Bornedal	107	horror + thriller
1995	Tokyo Fist	Shinya Tsukamoto	87	action + drama + thriller
1995	O Combate - Lágrimas do Guerreiro	Christophe Gans	102	action + crime + thriller
1995	Homem Morto	Jim Jarmusch	121	drama + fantasy + western
1995	Os 12 Macacos	Terry Gilliam	129	mystery + sci-fi + thriller
1996	Morte ao Vivo	Alejandro Amenábar	125	horror + mystery + thriller
1997	Jackie Brown	Quentin Tarantino	154	crime + drama + thriller
1999	O Sexto Sentido	M. Night Shyamalan	107	drama + mystery + thriller
1999	Quero Ser John Malkovich	Spike Jonze	112	comedy + fantasy + sci-fi

Below the table is a Facebook social plugin for comments. It includes a comment input field, a 'Comment using...' dropdown, and a list of comments from users: Eliana Boaglio, Fernando Boaglio, and Eduardo Cerqueira.

Figura 7.3: comentários de filme

7.3 INTEGRAÇÃO VIA SECURESOCIAL

Uma das coisas mais chatas que existem hoje é fazer cadastro de usuário e senha de site, e uma das coisas mais práticas inventadas nos últimos tempos é a possibilidade de autenticar-se em um site usando o login do Facebook, Google, Twitter, LinkedIn, e as demais redes sociais.

Felizmente, utilizando o plugin Secure Social (<http://securesocial.ws>) conseguiremos com poucos ajustes fazer essa integração no nosso site de filmes. O Secure Social é um módulo de autenticação do Play Framework que suporta os protocolos mais usados do mercado: OAuth, OAuth2, OpenID, usuário/senha e proporciona algumas informações dos usuários autenticados, como nome, sobrenome, nome completo e e-mail.

Vamos integrar da seguinte maneira: para cadastro e alteração de diretores, é necessário logar-se com o Facebook, e para listar os diretores aparecerá a mensagem

“bem-vindo” junto com o nome completo do usuário do Facebook. Na tela de lista de diretores, se o usuário estiver logado, aparecerá seu nome completo; caso contrário, aparecerá *guest*, como na figura 7.4.



Figura 7.4: Lista de diretores sem autenticação

Inicialmente configuramos o uso do módulo *Secure Social* no arquivo `build.sbt`:

```
libraryDependencies += Seq(  
  jdbc,  
  javaJdbc,  
  javaEbean,  
  cache,  
  "postgresql" % "postgresql" % "9.1-901-1.jdbc4",  
  "securesocial" %% "securesocial" % "2.1.2"  
)
```

Em seguida, vamos adicionar as novas rotas ao arquivo `routes.conf` como orienta o site do plugin (<http://securesocial.ws/guide/installation.html>).

```
# pagina de login  
  
GET    /login  
       securesocial.controllers.LoginPage.login  
  
GET    /logout  
       securesocial.controllers.LoginPage.logout  
  
# registro de usuario e gerenciamento de senha  
  
GET    /signup  
       securesocial.controllers.Registration.startSignUp
```

```

POST    /signup
        securesocial.controllers.Registration.handleStartSignUp
GET     /signup/:token
        securesocial.controllers.Registration.signUp(token)
POST    /signup/:token
        securesocial.controllers.Registration.handleSignUp(token)
GET     /reset
        securesocial.controllers.Registration.startResetPassword
POST    /reset
        securesocial.controllers.Registration.handleStartResetPassword
GET     /reset/:token
        securesocial.controllers.Registration.resetPassword(token)
POST    /reset/:token
        securesocial.controllers.Registration.handleResetPassword(token)
GET     /password
        securesocial.controllers.PasswordChange.page
POST    /password
        securesocial.controllers.PasswordChange.handlePasswordChange

# pontos de entrada dos provedores de autenticação

GET     /authenticate/:provider
        securesocial.controllers.ProviderController
                                .authenticate(provider)
POST    /authenticate/:provider
        securesocial.controllers.ProviderController
                                .authenticateByPost(provider)
GET     /not-authorized
        securesocial.controllers.ProviderController.notAuthorized

```

Depois, criamos um arquivo de plugins chamado `play.plugins` no diretório `conf`, que indica os plugins utilizados:

```

1500:com.typesafe.plugin.CommonsMailerPlugin
9994:securesocial.core.DefaultAuthenticatorStore
9995:securesocial.core.DefaultIdGenerator
9996:securesocial.core.providers.utils.DefaultPasswordValidator
9997:securesocial.controllers.DefaultTemplatesPlugin
9998:service.InMemoryUserService
9999:securesocial.core.providers.utils.BCryptPasswordHasher
10000:securesocial.core.providers.TwitterProvider
10001:securesocial.core.providers.FacebookProvider

```

```
10002:securesocial.core.providers.GoogleProvider
10003:securesocial.core.providers.LinkedInProvider
10004:securesocial.core.providers.UsernamePasswordProvider
10005:securesocial.core.providers.GitHubProvider
10006:securesocial.core.providers.FoursquareProvider
10007:securesocial.core.providers.XingProvider
10008:securesocial.core.providers.VkProvider
10009:securesocial.core.providers.InstagramProvider
```

O número ao lado do nome do plugin determina a sua prioridade, ou seja, a ordem em que ele será carregado (no nosso arquivo, o `CommonsMailerPlugin` será carregado primeiro).

Em seguida, adicionamos o arquivo `securesocial.conf`, que contém as configurações gerais do plugin: informações de e-mail e chaves dos provedores de autenticação das redes sociais.

Um exemplo desse arquivo está disponível no site <http://securesocial.ws/guide/configuration.html>, e nele vamos apenas alterar as configurações existentes do Facebook, adicionando a `clientId` e `clientSecret` de nossa aplicação:

```
facebook {
    authorizationUrl="https://graph.facebook.com/oauth/authorize"
    accessTokenUrl="https://graph.facebook.com/oauth/access_token"
    clientId=260229664128047
    clientSecret=1808b7b1d37ff5ca9bc95b21648a397e
    scope=email
}
```

Na sequência, colocaremos uma chamada a esse arquivo no final do `application.conf`:

```
# Logger provided to your application:
logger.application=DEBUG

include "securesocial.conf"
```

Encerrada a configuração, vamos aos ajustes em nosso código. Inicialmente, vamos alterar o controller `DiretorCRUD` para obter o nome do usuário do Facebook com a variável `userName`:

```
@SecureSocial.UserAwareAction
public static Result lista() {
```

Com a anotação `@SecureSocial.UserAwareAction`, permite-se obter o usuário do Facebook nesse método.

```
List<Diretor> diretores = Diretor.find.findList();
```

Aqui buscamos a lista de diretores, como vimos no capítulo de persistência 3.

```
Identity user = (Identity) ctx().args.get(SecureSocial.USER_KEY);
```

Essa é a chamada em que o plugin Secure Social busca as informações e atribui ao objeto `user`.

```
final String userName = user != null ? user.fullName() : "guest";
```

```
return ok(views.html.diretor.render(diretores, userName));
```

```
}
```

E finalmente o nome do usuário, se estiver logado, recebe o nome completo; caso contrário, recebe o valor “guest” e é atribuído à variável `userName`.

Em seguida nas rotinas de cadastrar e remover diretor, vamos adicionar a anotação apropriada para exigir autenticação no Facebook:

```
@SecureSocial.SecuredAction
public static Result remover(Long id) {
```

```
    ...
```

```
}
```

```
@SecureSocial.SecuredAction
public static Result novoDiretor() {
```

```
    ...
```

```
}
```

```
@SecureSocial.SecuredAction
public static Result alterar(Long id) {
```

```
    ...
```

```
}

@SecureSocial.SecuredAction
public static Result gravar() {

    ...

}
```

E finalmente, no template `diretor.scala.html`, mudamos os parâmetros de entrada, adicionando o `usuario`:

```
@(diretores: List[Diretor])(usuario: String)

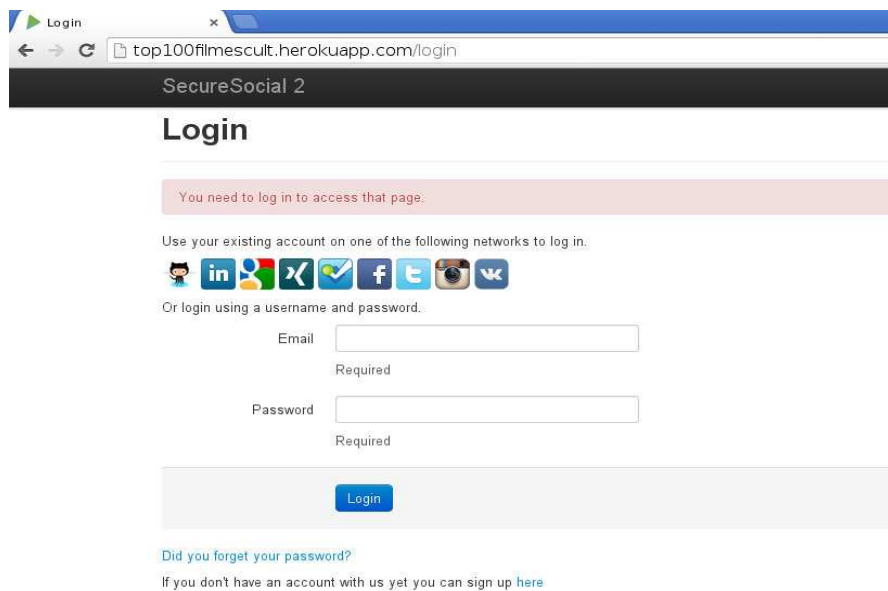
@main("Diretores") {

    <div class="container">
```

E em seguida, adicionamos uma mensagem de bem-vindo com o nome do usuário:

```
<div class="header">
  <ul class="nav nav-pills pull-right">
    <li class="active">
      <a href="@routes.DiretorCRUD.novoDiretor()">Novo diretor</a>
    </li>
    <li class="active"><a href="/">Home</a></li>
  </ul>
  <h2 class="text-muted">Bem-vindo @usuario ! </h2>
  <h3 class="text-muted">Diretores </h3>
</div>
```

Ao tentarmos acessar a tela de cadastro de novo diretor, o plugin redireciona para a tela de autenticação, como exibido na figura 7.5.



The screenshot shows a web browser window with the address bar displaying `top100filmscult.herokuapp.com/login`. The page title is "SecureSocial 2". The main heading is "Login". Below the heading, there is a red error message: "You need to log in to access that page." Underneath, it says "Use your existing account on one of the following networks to log in." and displays a row of social media icons: GitHub, LinkedIn, Google, X, Next.js, Facebook, Twitter, Instagram, and VK. Below the icons, it says "Or login using a username and password." and provides two input fields: "Email" and "Password". Both fields have a "Required" label below them. At the bottom of the form is a blue "Login" button. Below the button, there are two links: "Did you forget your password?" and "If you don't have an account with us yet you can sign up [here](#)".

Figura 7.5: Autenticação com Secure Social

No nosso primeiro acesso, o Facebook pergunta se é permitido o acesso dos dados do usuário pelo nosso site (figura 7.6).

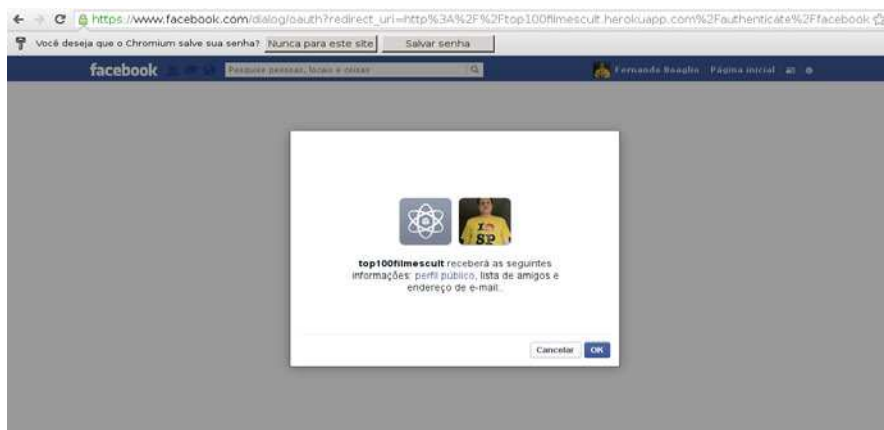


Figura 7.6: Permissão de acesso aos dados do usuário

Finalmente, depois de autenticado, o nome do usuário é exibido com sucesso, como mostra a figura 7.7.



Figura 7.7: Lista de diretores com usuário autenticado

Aqui concluímos que o plugin Secure Social permite facilmente obter informação do usuário logado e também restringir o acesso em algumas telas, possibilitando a criação de uma estrutura mais complexa de controle de permissão, e também, se necessário, armazenar essa informação no banco de dados.

7.4 PRÓXIMOS PASSOS

Certifique-se de que aprendeu:

- a cadastrar uma aplicação no Facebook;
- a adicionar chamada JavaScript para incluir comentários do Facebook;
- a integrar o login do Facebook utilizando o Secure Social.

Agora que já temos o site integrado às redes sociais, vamos adicionar algumas melhorias, como serviço de upload e alguns testes.

CAPÍTULO 8

Melhorias na aplicação

Nossa aplicação já cumpre o seu papel, mas existem pequenas melhorias que podem ser feitas. Nesse capítulo aprenderemos a ativar o HTTPS no site, fazer upload de imagens e criar testes efetivos de nosso sistema.

8.1 CONFIGURANDO HTTPS

Em alguns ambientes o uso de HTTPS é obrigatório, e nos ambientes de servidores Java tradicionais é necessário fazer uma série de passos para ter um ambiente HTTPS, como por exemplo, criar um certificado assinado localmente ou mudar as portas.

No Play, a criação de um ambiente assim é feita automaticamente passando apenas um parâmetro, que será a porta que será executado o HTTPS:

```
fb@cascao ~/workspace-play/filmes> play
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
```

```
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
```

```

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

```

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

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

```
[filmes] $ run -Dhttps.port=9443
```

```
--- (Running the application from SBT, auto-reloading is enabled) ---
```

```
[info] play - Listening for HTTP on /0.0.0.0:9000
[info] play - Listening for HTTPS on port /0.0.0.0:9443
```

```
(Server started, use Ctrl+D to stop and go back to the console...)
```

Note pelo log que o serviço HTTP subiu na porta 9000 e o HTTPS na 9443. Todo o processo de criar um certificado assinado localmente é feito pelo Play internamente.

Acessando o sistema com um browser, obtemos um aviso exibido na figura [8.1](#).



Figura 8.1: primeiro acesso em HTTPS

Esse aviso significa que o site que estamos visitando não é garantido por uma unidade certificadora, como por exemplo a empresa Verisign. Para adquirir um certificado digital, é preciso pagar para essas empresas, que funcionam semelhantes a cartórios virtuais, elas asseguram que o site que está sendo acessado é realmente da empresa que ele representa. Quando um certificado é assinado localmente (como no nosso caso), o site funciona, apenas exibindo um aviso. No caso do Google Chrome, o https aparece riscado em vermelho, como mostra a figura 8.2.

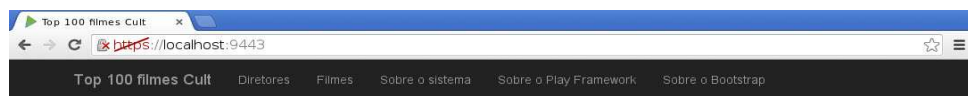


Figura 8.2: o sistema em HTTPS

Se desejarmos ativar exclusivamente a porta HTTPS, basta passarmos um parâmetro para desativar a porta HTTP:

```
fb@cascao ~/workspace-play/filmes > play
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
```

```

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

```
play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
http://www.playframework.com
```

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

```
[filmes] $ run -Dhttps.port=9443 -Dhttp.port=disabled
```

```
--- (Running the application from SBT, auto-reloading is enabled) ---
```

```
[info] play - Listening for HTTPS on port /0.0.0.0:9443
```

(Server started, use Ctrl+D to stop and go back to the console...)

Note pelo log que o serviço HTTP não subiu e temos apenas o HTTPS na porta 9443.

Se for necessário configurar um certificado digital oficial (pago) em um ambiente de produção, será necessário seguir a documentação do servidor de aplicação escolhido. No caso do servidor do Play 2, as instruções oficiais estão no endereço <http://www.playframework.com/documentation/2.2.x/ConfiguringHttps>.

8.2 LENDO CONSTANTES GLOBAIS

Com o Play conseguimos definir facilmente valores globais no arquivo `application.conf`, apenas declarando como se fossem valores de arquivo `properties`:

```
servidorDeDocumentos = "192.168.10.123"
```

E de dentro do código conseguimos ler o valor da constante:

```
String servidorDeDocumentos =  
    Play.application().configuration()  
        .getString("servidorDeDocumentos");
```

Além de ler uma constante do tipo `String`, podemos ler outros tipos também, como `booleano` (com `getBoolean`) ou `numérico` (com `getInt`).

8.3 UPLOAD DE IMAGEM

Quando pensamos em um sistema web, pensamos também em um cadastro de usuários. Se esse cadastro possuir uma foto de perfil (o que é bem provável), será necessária uma opção de upload de imagem, que é algo que o Play permite fazer facilmente, como veremos adiante.

Vamos criar uma opção de adicionar uma imagem para cada filme existente. No final da tela de lista de filmes, teremos uma opção de upload de imagem, onde o usuário escolhe o filme, a imagem, manda o arquivo e ele aparecerá automaticamente na lista.

A tela de cadastro será criada como na figura 8.3.

The screenshot shows a web application with a table of movies and a form below it. The table lists movies with their titles, directors, ratings, genres, and vote counts. The form below the table is titled 'Escolha o filme' and 'Escolha uma imagem para fazer upload:'. It includes a dropdown menu for selecting a movie, a file upload button, and an 'Enviar imagem' button. The form is highlighted with a red rectangle.

Ano	Título	Diretor	Classificação	Nota	Votos
1993	Amor à Queima-Roupa	Tony Scott	crime + thriller	nota: 7.9000001 com 124771 votos	
1994	Pulp Fiction: Tempo de Violência	Quentin Tarantino	crime + drama + thriller	nota: 9.0 com 836247 votos	
1994	Assassinos por Natureza	Oliver Stone	crime + drama	nota: 7.19999981 com 130772 votos	
1994	Na Roda da Fortuna	Joel Coen	comedy	nota: 7.30000019 com 53071 votos	
1994	Riget		comedy + drama + fantasy + horror + mystery	nota: 8.99999962 com 10311 votos	
1994	Nightwatch - Perigo na Noite	Ole Bornedal	horror + thriller	nota: 7.30000019 com 9806 votos	
1995	Homem Morto	Jim Jarmusch	drama + fantasy + western	nota: 7.5999999 com 55878 votos	
1995	O Combate - Lágrimas do Guerreiro	Christophe Gans	action + crime + thriller	nota: 6.30000019 com 7895 votos	
1995	Tokyo Fist	Shinya Tsukamoto	action + drama + thriller	nota: 7.0999999 com 1750 votos	
1995	Os 12 Macacos	Terry Gilliam	mystery + sci-fi + thriller	nota: 8.10000038 com 319642 votos	
1996	Morte ao Vivo	Alejandro Amenábar	horror + mystery + thriller	nota: 7.4000001 com 20229 votos	
1997	Jackie Brown	Quentin Tarantino	crime + drama + thriller	nota: 7.5 com 173812 votos	
1999	O Sexto Sentido	M. Night Shyamalan	drama + mystery + thriller	nota: 8.19999981 com 479507 votos	
1999	Quero Ser John Malkovich	Spike Jonze	comedy + fantasy + sci-fi	nota: 7.80000019 com 196825 votos	

Formulário de upload de imagem:

Escolha o filme: 2 - Psicose

Escolha uma imagem para fazer upload: Escolher arquivo Nenhum arquivo selecionado

Enviar imagem

Figura 8.3: cadastro de imagem de um filme

Inicialmente definimos o diretório de upload das imagens dos filmes no arquivo de configurações `application.conf`:

```
diretorioDeImagens = "/tmp/imagens/"
```

Em seguida vamos alterar o arquivo `filme.scala.html` para exibir a informação do filme e o campo para upload da imagem, adicionando o trecho a seguir no final do arquivo.

```
...
código da lista de filmes
...
</table>
<br/>
<br/>
@helper.form(action = routes.FilmeCRUD.upload(),
  'enctype -> "multipart/form-data") {
```

Iniciamos com a chamada à rotina de upload que adiciona o `enctype` do tipo `multipart/form-data`, que é obrigatório no formulário HTML para fazer upload.

```
<table class="table table-striped table-bordered"
      cellpadding="0" cellspacing="0" border="0" width="100%">
  <tr>
    <th>Escolha o filme</th>
    <td>
      <select name="filmeId" >
        @for(filme <- filmes) {
          <option value="@filme.id">@filme.id - @filme.nome</option>
        }
      </select>
    </td>
  </tr>
</table>
```

Depois adicionamos a lista de filmes e em seguida a opção de upload (`type="file"`) no parâmetro `picture`.

```
<tr>
  <th>Escolha uma imagem para fazer upload:</th>
  <td>
    <input type="file" name="picture" class="form-control">
  </td>
</tr>
<tr>
  <td colspan="2">
    <input type="submit" class="btn btn-primary"
      value="Enviar imagem">
  </td>
</tr>
</table>
}
</div>
}
```

Em seguida precisamos criar no controller o método que receberá a imagem e escreverá no diretório especificado.

O método `upload` é configurado para receber arquivo, e em seguida configuramos a variável `picture` para receber o arquivo.

```
public static Result upload() {

    MultipartFormData body = request().body().asMultipartFormData();
    FilePart picture = body.getFile("picture");
}
```


Se o usuário não enviou nenhum arquivo, ele virá nulo, portanto, precisamos fazer essa verificação antes de montar o nome do arquivo.

Se o arquivo não for nulo, buscamos também o ID do combo de filmes e adicionamos ao nome do arquivo: <ID-DO-FILME>-<ARQUIVO>.

```
if (picture != null) {

    String filmeId = form().bindFromRequest().get("filmeId");
    String imagem = filmeId + ".png";
```

Em seguida escrevemos o arquivo no diretório configurado na variável global `diretorioDeImagens`:

```
File file = picture.getFile();

String diretorioDeImagens =
    Play.application().configuration()
        .getString("diretorioDeImagens");

file.renameTo(new File(diretorioDeImagens, imagem));
```

E finalmente temos o redirecionamento para uma página `upload.scala.html`, em caso de sucesso, e para a página inicial em caso de erro.

```
return ok(views.html.upload.render("Arquivo \"" +
    imagem + "\" foi carregado com sucesso !"));

    } else {
        flash("error", "Erro ao fazer upload");
        return redirect(routes.Application.index());
    }
}
```

A página `upload.scala.html` é bem simples e apenas exibe uma mensagem e um link para retornar para a lista de filmes.

```
@(mensagem: String)

@main("Upload finalizado com sucesso") {

<div class="container">
```

```

<div class="header">
  <h2>@mensagem</h2>
</div>
<div class="control-group">
  <label class="control-label" for="singlebutton"></label>
  <div class="controls">
    <a href="/filme" class="btn btn-primary">
      retornar para lista de filmes
    </a>
  </div>
</div>
</div>
}

```

E para finalizar apenas precisamos adicionar a rota para o upload de imagem no arquivo de rotas:

```
POST    /filmesUpload      controllers.FilmeCRUD.upload()
```

Com isso faremos o upload da imagem `A.Clockwork.Orange.png`, conforme a figura 8.4.

Figura 8.4: cadastro de uma imagem

Depois do upload, a mensagem é exibida com sucesso e a imagem gravada no diretório especificado, conforme o exemplo adiante (figura 8.5).

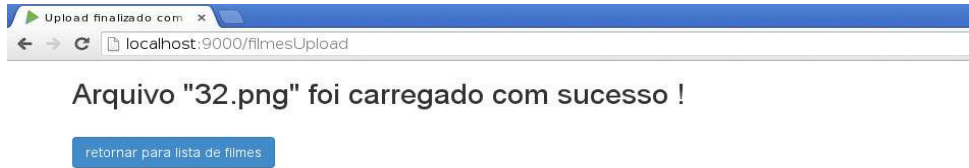


Figura 8.5: upload da imagem com sucesso

Vamos fazer um método para ler as imagens de que fizemos upload, inicialmente adicionamos a rota, o método `imagem` terá um parâmetro `id` que representa a identificação de um filme.

```
GET      /filmeImagem      controllers.FilmeCRUD.imagem(id:Long)
```

Para fazermos um teste rápido, vamos adicionar uma chamada da imagem de que fizemos o upload no início da lista de filmes no arquivo `filme.scala.html`:

```
@(filmes: List[Filme])
```

```
@main("Filmes") {
```

```

```

```
<div id="fb-root"></div>
```

O resultado esperado é a imagem aparecer no topo, conforme a figura 8.5.



Figura 8.6: imagem do upload carregada na tela

Percebemos que a rotina funciona, mas será mais adequado exibir a imagem ao lado do nome do filme. Portanto, no mesmo arquivo vamos substituir esse trecho:

```
<td>
  <a href="@routes.FilmeCRUD.detalhar(filme.id)">
    @filme.nome
  </a>
</td>
```

Por esse:

```
<td>
  <a href="@routes.FilmeCRUD.detalhar(filme.id)">@filme.nome</a>
  
</td>
```

E como resultado temos a imagem ao lado do filme corretamente (figura 8.7).



1969	A Besta Deve Morrer	A Besta Deve Morrer	Claude Chabrol	110	thriller	nota: 7.69999981 com 2185 votos
1969	Sem Destino	Sem Destino	Dennis Hopper	95	drama	nota: 7.30000019 com 58748 votos
1970	Zabriskie Point	Zabriskie Point	Michelangelo Antonioni	110	drama + romance	nota: 6.9000001 com 8000 votos
1971	James Wan	James Wan				nota: com votos
1971	Corrida Sem Fim	Corrida Sem Fim	Monte Hellman	102	drama	nota: 7.19999981 com 6256 votos
1971	Sob o Domínio do Medo	Sob o Domínio do Medo	Sam Peckinpah	118	drama + thriller	nota: 7.59999938 com 35931 votos
1971	A Clockwork Orange	A Clockwork Orange	Stanley Kubrick	136	crime + drama + sci-fi	nota: 8.39999962 com 359551 votos
1972	Solaris	Solaris	Andrei Tarkovsky	167	drama + sci-fi	nota: 8.0 com 36132 votos

Figura 8.7: imagem do upload carregada corretamente

Entretanto, a nossa aplicação tem um problema: ela permite o upload de qualquer tipo de arquivo e exibe apenas imagens do tipo PNG. Se escolher uma imagem `rosemary-baby.jpg`, ela será enviada com sucesso, mas não será exibida. Para solucionar esse problema, vamos limitar esse tipo de arquivo no upload, fazendo a seguinte alteração.

Inicialmente vamos adicionar novas variáveis globais no arquivo `application.conf`:

```
contentTypePadraoDeImagens="image/png"
extensaoPadraoDeImagens=.png
```

Em seguida, o novo código do método de upload.

Primeiro, lemos os valores das constantes globais.

```
public static Result upload() {

    MultipartFormData body = request().body().asMultipartFormData();
    FilePart picture = body.getFile("picture");
    String extensaoPadraoDeImagens =
        Play.application().configuration()
            .getString("extensaoPadraoDeImagens");
    if (picture != null) {
```

```
String filmeId = form().bindFromRequest().get("filmeId");
String imagem = filmeId + extensaoPadraoDeImagens;
```

Aqui usamos o método `getContentType` para identificar o tipo de imagem que está sendo enviada. No exemplo vamos tentar enviar uma imagem do tipo JPEG (figura 8.8).

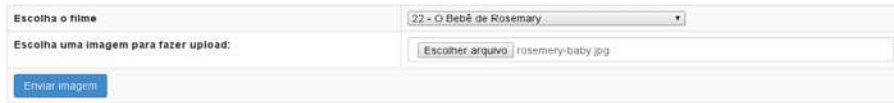


Figura 8.8: imagem do upload de arquivo do tipo JPEG

```
String contentType = picture.getContentType();
File file = picture.getFile();

String diretorioDeImagens =
    Play.application().configuration()
        .getString("diretorioDeImagens");
String contentTypePadraoDeImagens =
    Play.application().configuration()
        .getString("contentTypePadraoDeImagens");
```

Aqui identificamos se o tipo de arquivo enviado é o `image/PNG`. Em caso positivo, o comportamento é o mesmo do método anterior, e em caso negativo é exibida a mensagem de erro conforme a figura 8.9.

```
if (contentType.equals(contentTypePadraoDeImagens)) {

    file.renameTo(new File(diretorioDeImagens, imagem));
    return ok(views.html.upload.render("Arquivo \" +
        imagem + "\" do tipo [" + contentType +
        "] foi carregado com sucesso !"));

} else { // se for uma imagem nao aceita

    return ok(views.html.upload.render(
        "Imagens apenas no formato \"" +
        contentTypePadraoDeImagens + "\" serão aceitas!"));
```

```
    }  
  }  
  
  else {    // se o usuario nao enviou o arquivo  
  
    flash("error", "Erro ao fazer upload");  
    return redirect(routes.Application.index());  
  }  
}
```

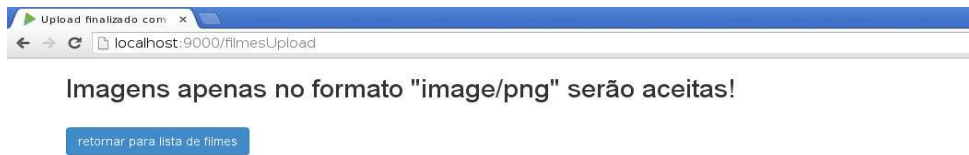


Figura 8.9: imagem de erro ao tentar o upload de arquivo do tipo JPEG

Com isso, conseguimos fazer o upload de imagens usando os recursos próprios do play.

8.4 TESTANDO SUA APLICAÇÃO

Conforme nossa aplicação vai crescendo, a chance de uma nova alteração afetar algo que já funciona aumenta, e para impedir que isso aconteça, é interessante ter uma boa cobertura de testes, principalmente nas partes críticas do sistema.

O Play nos ajuda nisso também, proporcionando diferentes tipos de teste.

Iniciaremos com um teste chamando um controller e testando o seu resultado.

Criaremos uma classe com um método `testaHTMLdoIndex` dentro do pacote `test`.

```
import static org.fest.assertions.Assertions.assertThat;  
import static play.mvc.Http.Status.OK;  
import static play.test.Helpers.*;  
  
import org.junit.Test;
```

```
import play.mvc.Result;

public class WebTest {

    @Test
    public void testaHTMLdoIndex() {
```

Aqui buscamos o resultado com o método `callAction` chamando o controller `Inicio.index()`. Em seguida, comparamos se o resultado é o status de sucesso (equivalente ao status HTTP 200 — http://pt.wikipedia.org/wiki/Anexo:Lista_de_c%C3%B3digos_de_status_HTTP).

```
Result result = callAction(controllers.routes.ref.Inicio.index());
assertThat(status(result)).isEqualTo(OK);
```

Com isto comparamos se o resultado retornou um arquivo do tipo texto HTML:

```
assertThat(contentType(result)).isEqualTo("text/html");
```

Depois comparamos se o encoding retornado é UTF-8:

```
assertThat(charset(result)).isEqualTo("utf-8");
```

E finalmente verificamos se no resultado existe o texto “melhor do cinema”.

```
assertThat(contentAsString(result)).contains("melhor do cinema");
    }
}
```

Para iniciar uma aplicação do play já aprendemos que o comando é `play run`. Já para rodar os testes o comando é semelhante: `play test`.

```
fb@cascao ~/workspace-play/filmes > play test
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
[info] WebTest
[info] + testaHTMLdoIndex
[info]
[info]
[info] Total for test WebTest
```



```
[info] Finished in 0.02 seconds
[info] 1 tests, 0 failures, 0 errors
[info] Passed: Total 1, Failed 0, Errors 0, Passed 1
[success] Total time: 2 s, completed 30/01/2014 22:56:08
fb@cascao ~/workspace-play/filmes >
```

No final do log temos `success`, que demonstra que o teste executou com sucesso.

TESTES NO ECLIPSE

Apesar de existir uma interface gráfica para executar testes no Eclipse, nem todo teste do Play é executado com sucesso, por esse motivo é imprescindível que toda execução de testes seja feita no play console.

Vamos ver como o teste se comporta em caso de falha, para isso basta trocarmos o texto:

```
assertThat(contentAsString(result)).contains("melhor do cinema");
```

Por:

```
assertThat(contentAsString(result))
    .contains("melhor do Fernando Boaglio");
```

Temos um log no console, bem maior que o anterior, executando o mesmo comando `play test`:

```
fb@cascao ~/workspace-play/filmes > play test
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
[info] Compiling 1 Java source to
/home/fb/workspace-play/filmes/target/scala-2.10/test-classes...
```

Aqui é detalhado o erro em `WebTest.testaHTMLdoIndex`:

```
[error] Test WebTest.testaHTMLdoIndex failed: <'
[error]
[error] <!DOCTYPE html>
```

```
[error]
[error] <html>
[error]   <head>
[error]     <title>Top 100 filmes Cult</title>
[error]     <link href="/assets/bootstrap/css/bootstrap.min.css"
[error]       rel="stylesheet" media="screen">
[error]     <link rel="stylesheet" media="screen"
[error]       href="/assets/stylesheets/main.css">
[error]     <link rel="shortcut icon" type="image/png"
[error]       href="/assets/images/favicon.png">
[error]     <script src="/assets/javascripts/jquery-1.9.0.min.js"
[error]       type="text/javascript"></script>
[error]   </head>
[error]   <body>
[error] <style>
[error] body {
[error]   padding-top: 50px;
[error] }
[error] .starter-template {
[error]   padding: 40px 15px;
[error]   text-align: center;
[error] }
[error] </style>
[error] <div class="navbar navbar-inverse navbar-fixed-top">
[error]   <div class="container">
[error]     <div class="navbar-header">
[error]       <button type="button" class="navbar-toggle"
[error]         data-toggle="collapse"
[error]         data-target=".navbar-collapse">
[error]
[error]     </button>
[error]     <a class="navbar-brand" href="/">
[error]       Top 100 filmes Cult
[error]     </a>
[error]   </div>
[error]   <div class="collapse navbar-collapse">
[error]     <ul class="nav navbar-nav">
[error]       <li><a href="/diretor">Diretores</a></li>
[error]       <li><a href="/filme">Filmes</a></li>
[error]       <li><a href="/sobre">Sobre o sistema</a></li>
[error]       <li><a href="/play">Sobre o Play Framework</a></li>
```

```

[error]          <li><a href="http://getbootstrap.com/">
                Sobre o Bootstrap</a></li>
[error]      </ul>
[error]  </div>
[error] </div>
[error] </div>
[error] <div class="container">
[error]   <div class="starter-template">
[error]     <h1>Top 100 filmes Cult</h1>
[error]     <p class="lead">O melhor do cinema est&aacute; aqui!</p>
[error]   </div>
[error] </div>
[error] </body>
[error] </html>
[error]

```

Depois de exibir todo o resultado, o log informa que o ele deveria conter o texto “melhor do Fernando Boaglio”, exibindo um `x` ao lado do teste com erro.

```

[error] '> should contain the String:<'melhor do Fernando Boaglio'>
[info] WebTest
[info] x testaHTMLdoIndex
[info]
[info]
[info] Total for test WebTest
[info] Finished in 0.014 seconds
[info] 1 tests, 1 failures, 0 errors
[error] Failed: Total 1, Failed 1, Errors 0, Passed 0
[error] Failed tests:
[error]       WebTest
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 3 s, completed 30/01/2014 23:05:27
fb@cascao ~/workspace-play/filmes >

```

Podemos testar qualquer coisa, inclusive manipular o banco de dados. Entretanto, provavelmente cadastrar informações no banco de dados vai atrapalhar o desenvolvimento, principalmente se ele for compartilhado com uma equipe.

Por esse motivo, é interessante ter configurações específicas de teste, como por exemplo, acessar um banco de dados diferente.

Vamos mostrar um exemplo em que cadastramos um diretor e buscamos um filme utilizando configurações específicas.

Inicialmente criaremos um arquivo `testes.conf` no mesmo diretório do `application.conf`, com o seguinte conteúdo:

```
#
# ambiente de testes
#
application.langs="en,pt"
ebean.default="models.*"
evolutionplugin=disabled
db.default.driver=org.postgresql.Driver
db.default.url="postgres://postgres:postgres@localhost/filmes"
logger.root=ERROR
logger.play=INFO
logger.application=DEBUG
diretorioDeImagens = "/tmp/testes/imagens/"
contentTypePadraoDeImagens="image/png"
extensaoPadraoDeImagens=.png
```

Nesse arquivo estamos usando o mesmo banco de dados, mas podemos alterar esse arquivo sem problemas e sem comprometer a aplicação.

Em seguida, criaremos o teste `DatabaseTest` com o método `startApp` e a anotação `@BeforeClass`, que significa que esse método será executado antes de chamarmos os métodos com `@Test`.

O método `startApp` é responsável por carregar as configurações com `ConfigFactory.parseFile`.

```
public class DatabaseTest {

    private static Configuration additionalConfigurations;

    public static play.test.FakeApplication app;

    @BeforeClass
    public static void startApp() {

        Config additionalConfig =
            ConfigFactory.parseFile(new File("conf/testes.conf"));
        additionalConfigurations = new Configuration(additionalConfig);
        System.out.println(additionalConfigurations.asMap());

        app = Helpers.fakeApplication(additionalConfigurations.asMap());
```

```
        Helpers.start(app);  
    }
```

Aqui, adicionamos o método que finaliza as configurações adicionais, executado depois de chamarmos os métodos com `@Test`.

```
@AfterClass  
public static void stopApp() {  
    Helpers.stop(app);  
}
```

Depois, o teste adiciona um diretor chamado “Fernando Meireles” e em seguida verifica se o id gerado não é nulo.

```
@Test  
public void testaDiretor() {  
  
    Diretor diretor = new Diretor();  
    diretor.nome = "Fernando Meireles";  
    diretor.save();  
  
    assertThat(diretor.id).isNotNull();  
  
}
```

Então, o teste busca um filme pelo id 32 e confere se o nome é “A Clockwork Orange”.

```
@Test  
public void testaFilme() {  
  
    Filme filme = Filme.find.byId(321);  
    assertThat("A Clockwork Orange").isEqualTo(filme.nome);  
  
}  
  
}
```

Agora rodamos os testes novamente e verificamos que, depois da compilação, são executados com sucesso.

```
fb@cascao ~/workspace-play/filmes > play test
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes
(in build file:/home/fb/workspace-play/filmes/)
[info] Compiling 2 Java sources to
    /home/fb/workspace-play/filmes/target/scala-2.10/test-classes...
[info] WebTest
{extensaoPadraoDeImagens=.jpg,
  db={default={driver=org.postgresql.Driver,
url=postgres://postgres:postgres@localhost/filmes}},
  application={langs=en,pt},
  evolutionplugin=disabled, logger={application=DEBUG,
  root=ERROR, play=INFO},
  ebean={default=models.Diretor,models.Filme},
  diretorioDeImagens=/tmp/imagens/,
  contentTypePadraoDeImagens=image/jpeg}
[info] + testaHTMLdoIndex
[info]
[info]
[info] Total for test WebTest
[info] Finished in 0.011 seconds
[info] 1 tests, 0 failures, 0 errors
```

Depois do primeiro teste encerrado, inicia-se o segundo:

```
[info] application - [securesocial] loaded templates plugin:
    securesocial.controllers.DefaultTemplatesPlugin
[info] play - Starting application default Akka system.
[debug] application - [securesocial] calling deleteExpiredTokens()
[info] application - [securesocial] loaded user service:
    class service.InMemoryUserService
[info] application - [securesocial] loaded password hasher bcrypt
[info] application - [securesocial] loaded identity provider: twitter
[info] application - [securesocial] loaded identity provider: facebook
[info] application - [securesocial] loaded identity provider: google
[info] application - [securesocial] loaded identity provider: linkedin
[info] application - [securesocial] loaded identity provider: userpass
[info] application - [securesocial] loaded identity provider: github
[info] application - [securesocial] loaded identity provider: foursquare
[info] application - [securesocial] loaded identity provider: xing
[info] application - [securesocial] loaded identity provider: vk
```

```
[info] application - [securesocial] loaded identity provider: instagram
[info] application - [securesocial] unloaded ... provider: instagram
[info] application - [securesocial] unloaded ... provider: vk
[info] application - [securesocial] unloaded ... provider: xing
[info] application - [securesocial] unloaded ... provider: foursquare
[info] application - [securesocial] unloaded ... provider: github
[info] application - [securesocial] unloaded ... provider: userpass
[info] application - [securesocial] unloaded ... provider: linkedin
[info] application - [securesocial] unloaded ... provider: google
[info] application - [securesocial] unloaded ... provider: facebook
[info] application - [securesocial] unloaded ... provider: twitter
[info] application - [securesocial] unloaded password hasher bcrypt
[info] play - Shutdown application default Akka system.
[info] DatabaseTest
[info] + testaDiretor
[info] + testaFilme
[info]
[info]
[info] Total for test DatabaseTest
[info] Finished in 0.001 seconds
[info] 2 tests, 0 failures, 0 errors
```

Aqui temos o segundo teste encerrado, e finalmente, o resumo final:

```
[info] Passed: Total 3, Failed 0, Errors 0, Passed 3
[success] Total time: 4 s, completed 30/01/2014 23:20:27
fb@cascao ~/workspace-play/filmes >
```

TESTE EM WEB COM SELENIUM

O Selenium é uma ferramenta de automação de testes funcionais para interfaces Web. Sua principal vantagem é a utilização do próprio navegador web para realização dos testes. Consulte o site oficial para mais informações: <http://www.seleniumhq.org/>.

Se quisermos fazer um teste com o browser, podemos usar o Selenium e felizmente o Play encapsula as principais funções, facilitando muito a construção de testes.

Vamos fazer um teste com resultado semelhante ao primeiro, em que verificamos se a página inicial contém o texto “melhor do cinema”.

Primeiro, configuramos direto no código um servidor do Play na porta 3333, que utilizará um banco de dados em memória (ou seja, qualquer alteração nele será desprezada).

Para mais detalhes nas configurações, consulte a documentação em <http://www.playframework.com/documentation/2.2.1/JavaFunctionalTest>.

```
public class BrowserTest {

    @Test
    public void testarIndex() {
        running(testServer(3333, fakeApplication(inMemoryDatabase())),
            HTMLUNIT,
            new Callback<TestBrowser>() {
```

Aqui chamamos o browser no servidor na porta 3333 e chamamos a URL <http://localhost:3333/>. Em seguida, é verificado se no código fonte da página foi retornado o texto “melhor do cinema”.

```
        public void invoke(TestBrowser browser) {
            browser.goTo("http://localhost:3333/");
            assertThat(browser.pageSource())
                .contains("melhor do cinema");
        }
    });
}
```

O interessante da API do Selenium é que ela permite reproduzir a navegação no sistema como se fosse o web browser.

Nesse exemplo, vamos acessar a página de listagem de filmes:

```
@Test
public void testarLinkNovoFilme() {
    running(testServer(3333, fakeApplication()),
        HTMLUNIT,
        new Callback<TestBrowser>() {

        public void invoke(TestBrowser browser) {
            browser.goTo("http://localhost:3333/filme");
```

Em seguida verificamos se ela contém o texto “Filmes”:

```
assertThat(browser.pageSource()).contains("Filmes");
```


Depois disso, com o método `click` simulamos o evento de o usuário clicar no primeiro link da lista de filmes, que nos levará para a página do filme Psicose.

O teste termina verificando se no código fonte HTML da página existe o texto “Psicose”.

```
browser.$("table td a").first().click();
    assertThat(browser.pageSource()).contains("Psicose");
  }
});
}
```

Aqui, finalmente, temos a execução completa: sempre que for executado o comando `play test`, todos os testes que criarmos serão executados.

```
fb@cascao ~/workspace-play/filmes > play test
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] WebTest
[info] + testaHTMLdoIndex
[info]
[info]
[info] Total for test WebTest
[info] Finished in 0.014 seconds
[info] 1 tests, 0 failures, 0 errors
[info] DatabaseTest
[info] + testaDiretor
[info] + testaFilme
[info]
[info]
[info] Total for test DatabaseTest
[info] Finished in 0.001 seconds
[info] 2 tests, 0 failures, 0 errors
[info] BrowserTest
[info] + testarLinkNovoFilme
[info] + testarIndex
[info]
[info]
[info] Total for test BrowserTest
[info] Finished in 0.0 seconds
[info] 2 tests, 0 failures, 0 errors
```

```
[info] Passed: Total 5, Failed 0, Errors 0, Passed 5  
[success] Total time: 19 s, completed 31/01/2014 00:34:54  
fb@cascao ~/workspace-play/filmes >
```

8.5 PRÓXIMOS PASSOS

Certifique-se de que aprendeu a:

- configurar HTTPS de uma aplicação do Play;
- fazer upload de arquivos;
- manipular arquivos para exibir no web browser;
- criar testes do controller;
- criar testes do banco de dados com configurações específicas;
- criar testes com Selenium.

CAPÍTULO 9

Continue seus estudos

Nossa aplicação está completa, mas com certeza pode ser melhorada.

A seguir algumas ideias do que poderia ser implementado:

- enviar emails dos filmes preferidos;
- integrar em outras redes sociais além do Facebook;
- criar uma área de usuário, com perfil, informações e fotos, filmes preferidos.

9.1 PARA SABER MAIS

Agora em diante, para aprimorar os conhecimentos no Play:

- Acompanhe a comunidade do Play Framework no Google Plus: <https://plus.google.com/communities/116192785110716864793>
- Leia a documentação do site: <http://www.playframework.com/documentation/2.2.x/Home>

- Participe do grupo de email <https://groups.google.com/forum/#!forum/play-framework>, escrevendo no título do email [play-2.2-java]
- Se quiser se aprofundar em Scala para entender melhor o fonte do Play, faça o curso gratuito online do Martin Odersky, um dos desenvolvedores da linguagem: <https://www.coursera.org/course/progfun>

E acompanhe os principais blogs:

- blog oficial: <http://typesafe.com/blog/PlayFramework>
- blog de James Roper, commiter da equipe do Play - <http://jazzy.id.au/default/tags/play>
- blog de Pascal Voitot, commiter da equipe do Play - <http://mandubian.com/>
- blog de James Ward, commiter da equipe do Play - <http://www.jamesward.com/category/play-framework/>
- blog com vários artigos de Play - <http://www.scoop.it/t/playframework>

CAPÍTULO 10

Apêndice A - Play console em detalhes

Aqui segue um breve resumo do console do play, listando alguns comandos que serão úteis com o uso no dia a dia.

Os comandos aqui listados são executados dentro do console, conforme o exemplo:

```
fb@cascao ~/workspace-play/filmes> play
[info] Loading project definition from
/home/fb/workspace-play/filmes/project
[info] Set current project to filmes (in build file:
/home/fb/workspace-play/filmes/)
```

```

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

play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_51),
<http://www.playframework.com>

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

[filmes] \$

- `help e tasks` - exibe os comandos existentes do console
- `compile` - recompila os fontes
- `clean` - limpa temporários e cache
- `run` - inicia o servidor
- `settings` - lista as variáveis do servidor
- `reload` - recarrega os arquivos do projeto
- `session list` - lista informações das sessões existentes

CAPÍTULO 11

Apêndice B - Instalação e configuração do PostgreSQL

Para conseguirmos persistir as informações do nosso site, precisamos de um banco de dados instalado para funcionar o ambiente do capítulo 3.

Inicialmente acessamos o site oficial <http://www.postgresql.org/download/> e baixamos versão mais recente disponível da série 9.x do PostgreSQL.

Vamos utilizar como exemplo a versão 9.3.1 para Windows.

Faça o download do pacote para Windows e execute-o com um usuário com privilégios de administrador da máquina.

Após descompactar em uma área temporária, será exibida a tela inicial conforme a figura 11.1 .



Figura 11.1: início da instalação

Na figura 11.2 informe o diretório de instalação (pode ser o valor padrão mesmo).

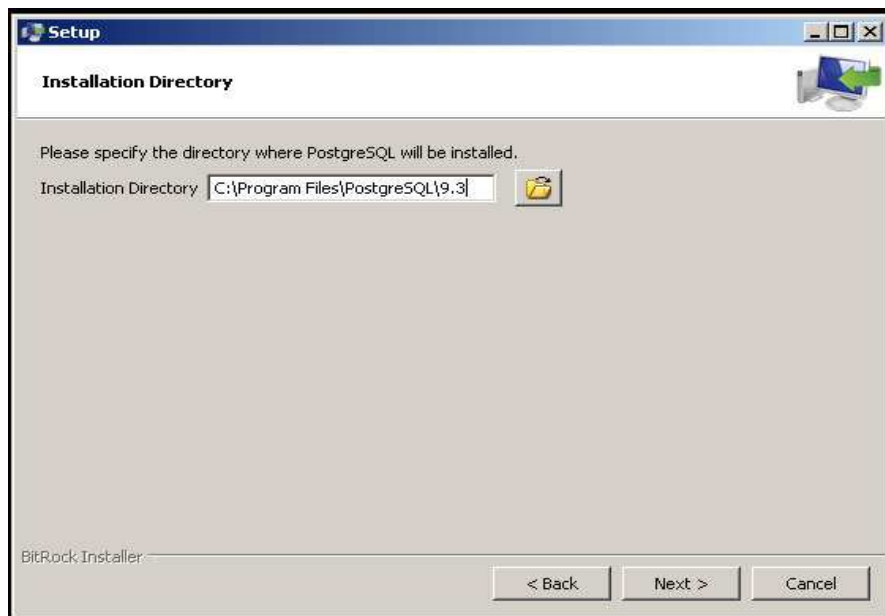


Figura 11.2: diretório de instalação

A instalação é feita sobre o usuário chamado `postgres`, forneça sua senha (também `postgres`) conforme a figura 11.3.

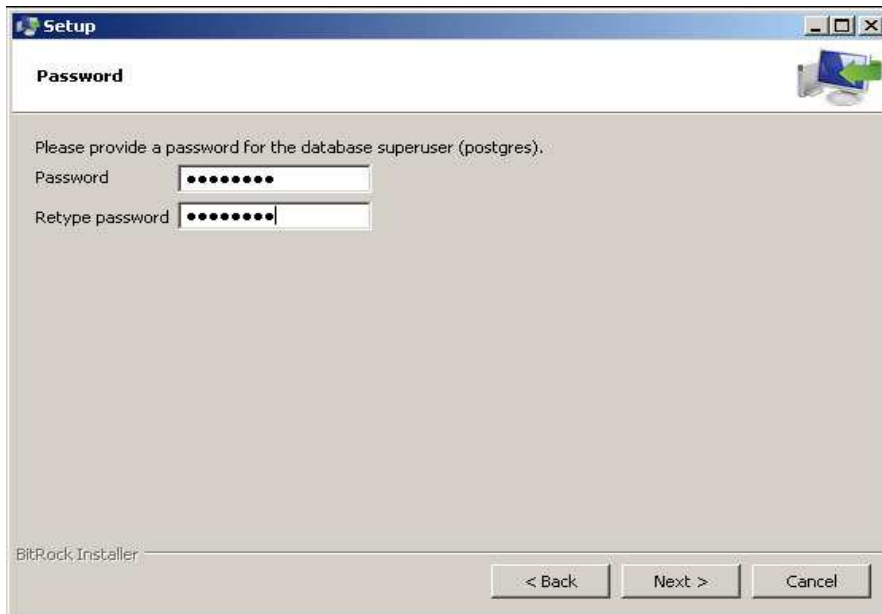


Figura 11.3: senha do usuário postgres

Continue a instalação até o final. O banco de dados será instalado e iniciado automaticamente. Além dele, foi instalada a ferramenta `pgAdmin 3`, conforme a figura 11.4.

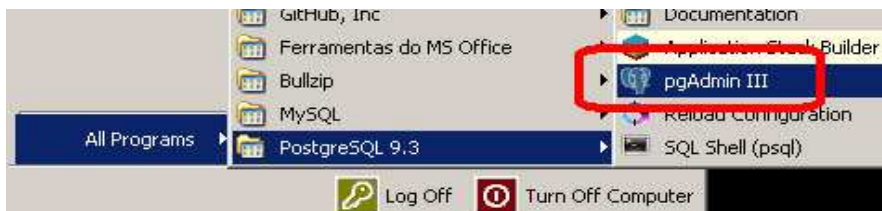


Figura 11.4: acesso ao pgAdmin 3

O `pgAdmin 3` é uma ferramenta completa de manutenção e manipulação do banco de dados do PostgreSQL.

Ao clicar na opção `PostgreSQL 9.3 (localhost:5432)` aparecerá uma janela pedindo a senha. Informe o valor `postgres` e marque a opção `Store password` como na figura 11.5.

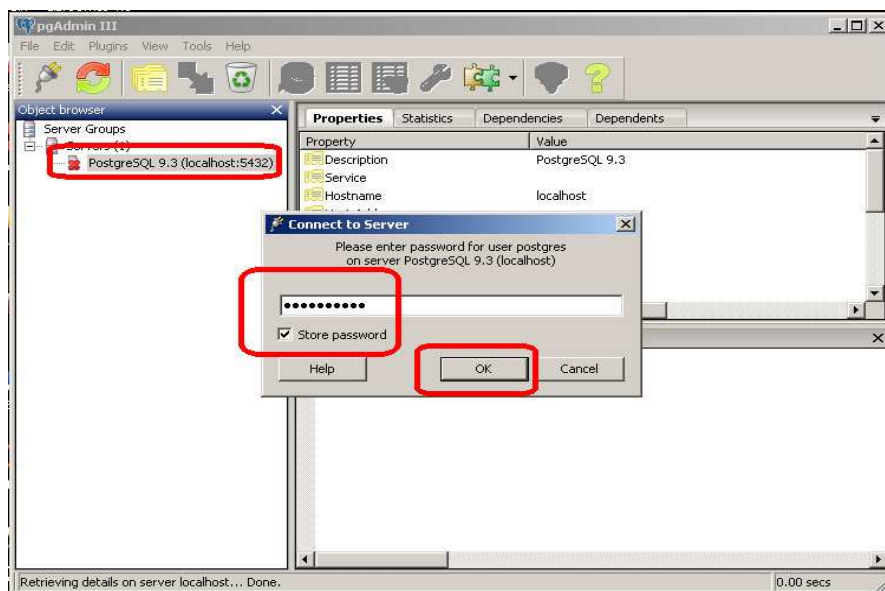


Figura 11.5: conectando ao banco de dados

Na opção `Database`, selecione a opção `New Database . . .` conforme a figura 11.6.

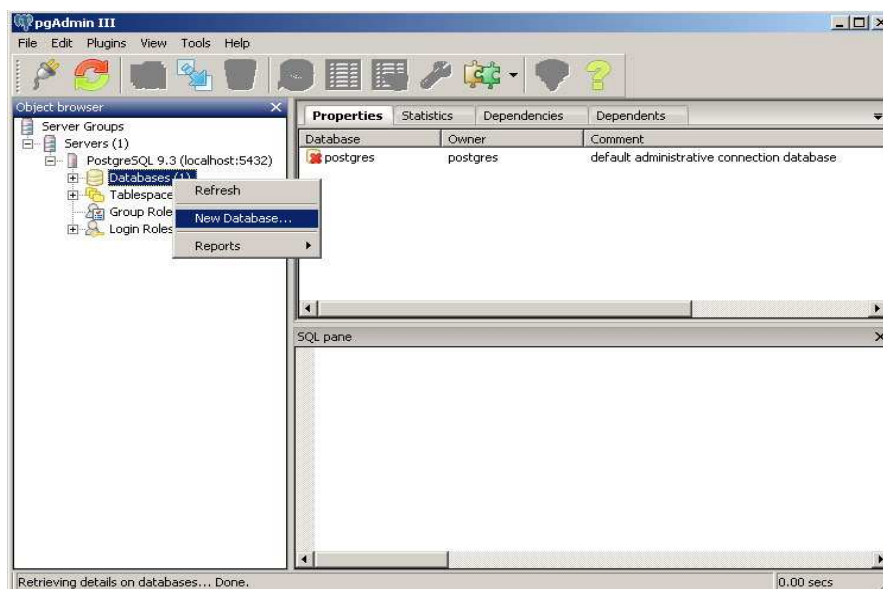


Figura 11.6: criando um novo banco de dados

No campo `name` é colocado o nome do banco de dados como na figura 11.7.

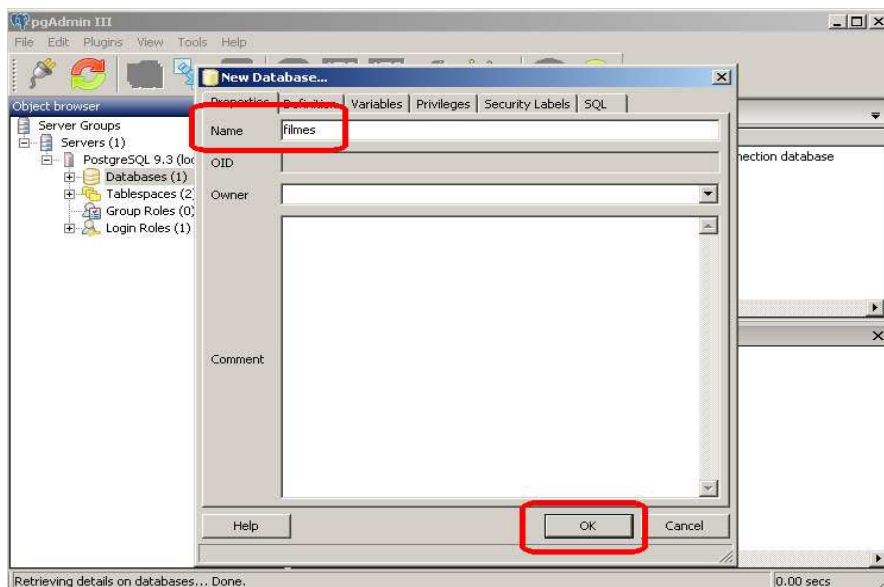


Figura 11.7: criando o banco de dados filmes

Depois de criar o banco filmes, selecione-o e clique no ícone com a lupa SQL conforme indicado na figura 11.8.

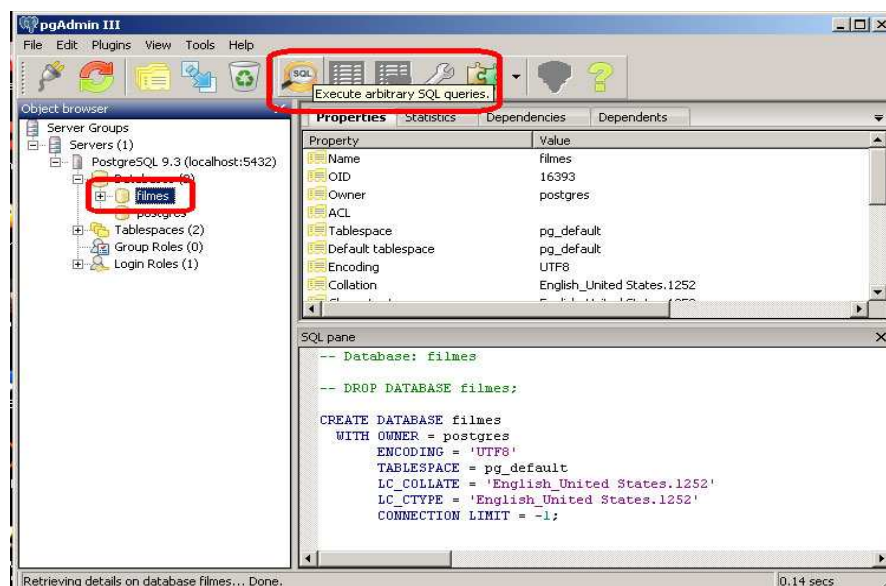


Figura 11.8: banco filmes criado

Acesse o SQL das tabelas pelo link <http://bit.ly/JLFA9C> e copie o seu conteúdo.

Em seguida cole dentro da janela SQL e clique onde está indicado na figura 11.9, e em seguida perceba a mensagem de execução com sucesso.

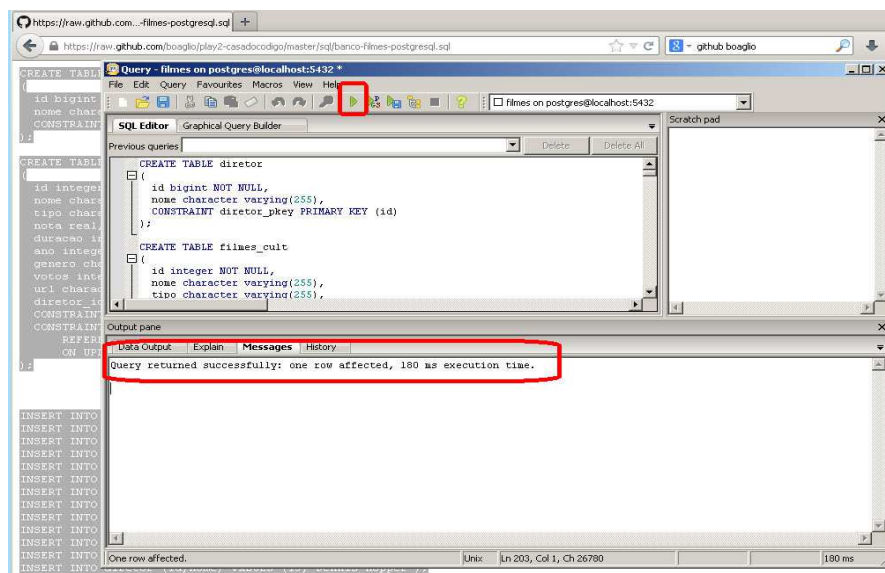


Figura 11.9: populando a base dados

As tabelas foram criadas com sucesso.

Clique na tabela e em seguida no ícone indicado na figura 11.10 para ver todos os registros.

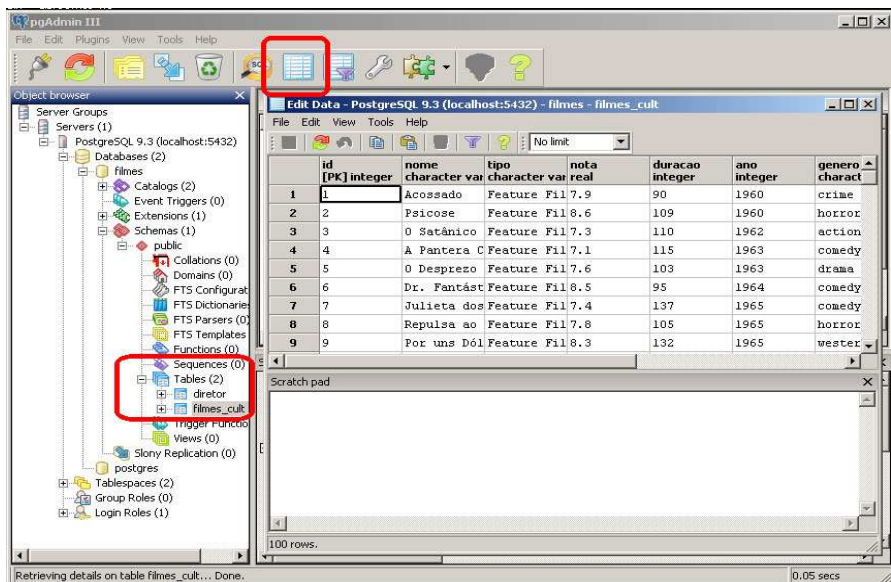


Figura 11.10: lista de filmes

O seu ambiente de banco de dados está pronto para uso, se necessário pode criar novos bancos de dados para testes sem problemas.

CAPÍTULO 12

Apêndice C - Instalação e configuração do Android

Esse apêndice é uma breve introdução ao Android com conceitos suficientes para executar o exemplo `play2-android-casadocodigo`, disponível em <https://github.com/boaglio/play2-android-casadocodigo>.

No capítulo de serviços (6) aprendemos a disponibilizar informações em formato JSON. A nossa aplicação em Android carregará essas informações nesse formato e exibirá na tela do celular.

Para desenvolver com Android, além da JDK serão necessárias mais duas ferramentas: Android SDK e o Eclipse com o ADT Plugin (chamado de ADT Bundle).

Baixe e instale o ADT bundle da sua plataforma <http://developer.android.com/sdk/index.html>.

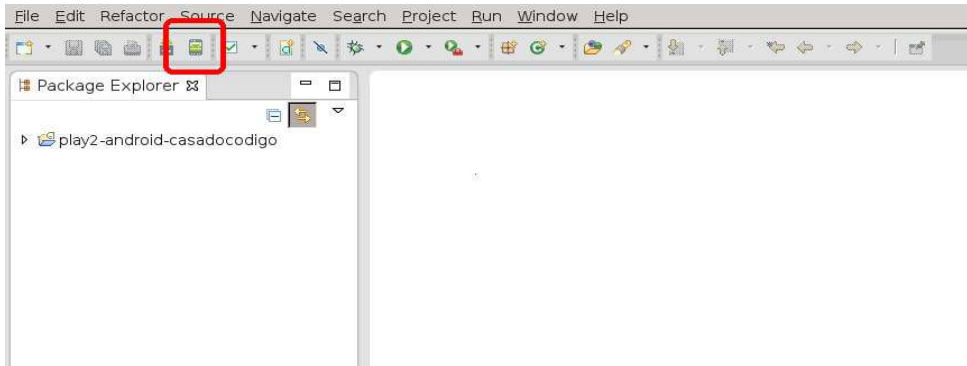
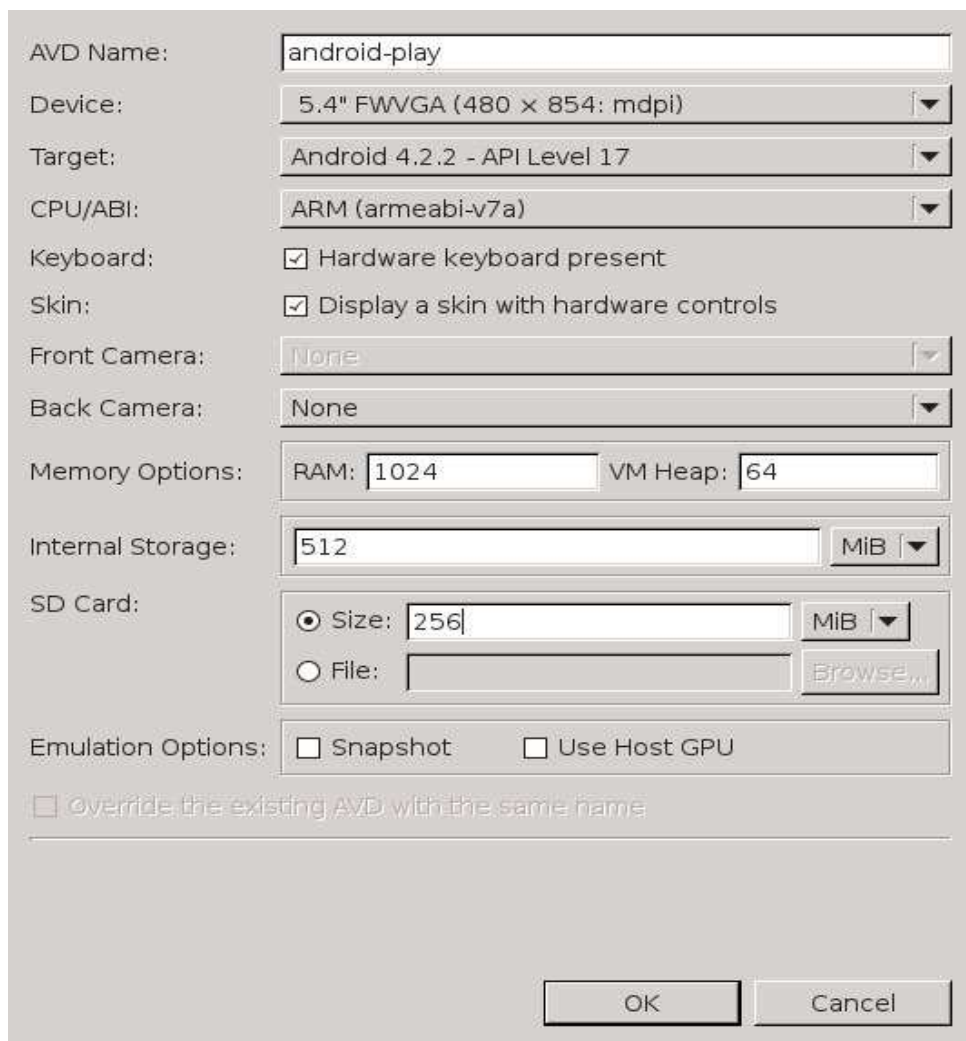


Figura 12.1: ADT bundle

Depois de importar o nosso projeto Android, clique no AVDM (Android Virtual Device Manager) como indicado na figura 12.1, que é o local que criamos emuladores de celular para rodar a nossa aplicação.



AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin: ☒ Display a skin with hardware controls

Front Camera:

Back Camera:

Memory Options: RAM: VM Heap:

Internal Storage:

SD Card: ☒ Size: ☐ File:

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Figura 12.2: emulador Android

Clicando em *New*, vamos criar um emulador conforme a figura 12.2.

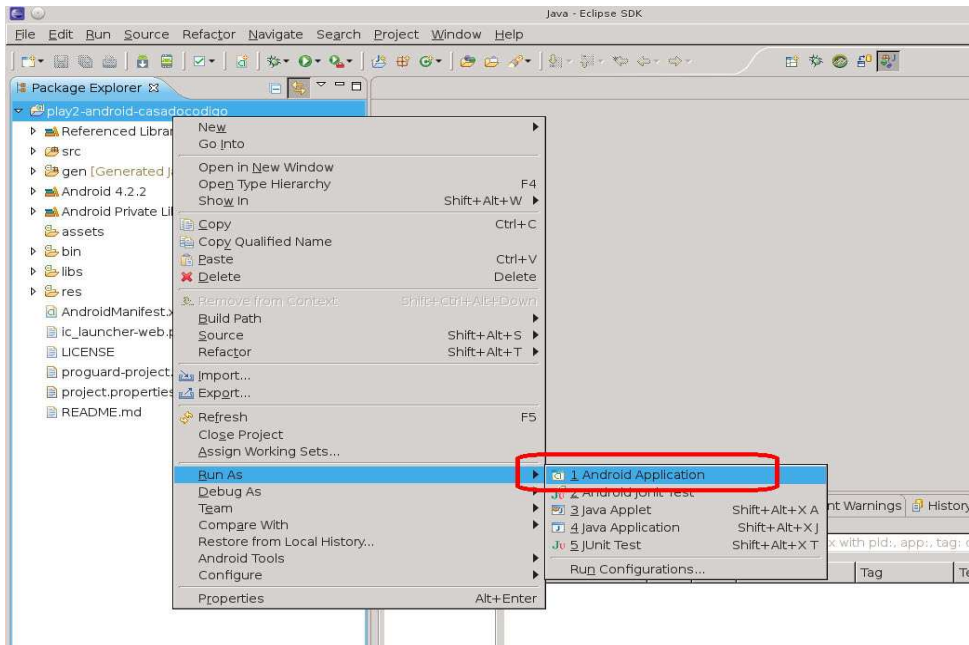


Figura 12.3: Executando aplicação Android

Agora com um emulador existente, chamando a opção `Run As` e `Android Application` como na figura 12.3, o emulador será executado automaticamente, e em seguida a aplicação será chamada.

Ao clicarmos no botão `buscar filmes`, a lista de filmes acessa a rotina criada no capítulo de serviços 6.3.



Figura 12.4: lista de filmes no Android

E temos como resultado a figura 12.4 exibindo a lista de filmes.