UNIVERSIDADE FEDERAL DA GRANDE DOURADOS

Neylson de Oliveira Gularte

Plataforma para Ensino de Desenvolvimento de Aplicações Web

DOURADOS

2016

Neylson de Oliveira Gularte

Plataforma para Ensino de Desenvolvimento de Aplicações Web

Trabalho de Conclusão de Curso de graduação apresentado para obtenção do título de Bacharel em Sistemas de Informação pela Faculdade de Ciências Exatas e Tecnologia da Universidade Federal da Grande Dourados.

Orientador: Prof.° Dr. Joinvile Batista Junior

DOURADOS

2016

Neylson de Oliveira Gularte

Plataforma para Ensino de Desenvolvimento de Aplicações Web

Trabalho de Conclusão de Curso aprovado como requisito para obtenção do título de Bacharel em Sistemas de Informação na Universidade Federal da Grande Dourados, pela comissão formada por:

_____

Orientador Prof. Dr. Joinvile Batista Junior
FACET – UFGD

_____

Prof. Me. Anderson Bessa da Costa
FACET – UFGD

_____

Prof. Dr. Wellington Lima dos Santos
FACET – UFGD

Dourados, 23 de setembro de 2016.

RESUMO

Neste trabalho é apresentado o modelo e implementação de uma plataforma para hospedagem de aplicações web feitas em Java para ser utilizada no ensino de desenvolvimento de aplicações web. É comum nos laboratórios de curso de graduação na área de computação a existência de restrições de acesso de conexões entre as máquinas dos alunos e instabilidades nas redes sem fio. Este tipo de restrição acaba limitando a demonstração de aplicações Web em localhost, quando o ideal seria pelo menos a conexão entre três máquinas, para demonstrar a distribuição do sistema web em três camadas: camada de apresentação (máquina do usuário), camada de negócios (servidor web) e camada de persistência (servidor com o sistema gerenciador de banco de dados). Em conjunto com a plataforma de hospedagem o trabalho também tem por objetivo apresentar uma alternativa para superar essa limitação com a criação de uma rede local sem fio entre notebooks sem a necessidade de hardware adicional. Permitindo dessa forma demonstrar a distribuição de um sistema web em três camadas. O presente trabalho utilizou, para o desenvolvimento de suas aplicações, uma arquitetura proposta para um Projeto de Extensão (BATISTA, 2014), baseada no modelo objeto relacional construído a partir das seguintes tecnologias: JavaServerFaces, EJB e Java Persistence API.

Palavras-Chave: aplicações web, plataforma de hospedagem, aplicações distribuídas.

SUMÁRIO

1 INTRODUÇÃO.....8

1.1 HISTÓRICO E MOTIVAÇÃO.....8

1.2 OPORTUNIDADES E RELEVÂNCIA.....9

1.3 OBJETIVOS DO TRABALHO.....10

1.4 METODOLOGIA ADOTADA.....10

1.5 CONTEÚDO DO TRABALHO.....11

2 FUNDAMENTAÇÃO TEÓRICA.....12

2.1 ARQUITETURA DO SISTEMA.....12

2.2 CRIAÇÃO DE REDE LOCAL SEM FIO.....14

2.3 SERVIDOR DE APLICAÇÕES WILDFLY.....16

3 DESENVOLVIMENTO DO TRABALHO PROPOSTO.....18

3.1 DESENVOLVIMENTO E EXPLICAÇÃO DO FUNCIONAMENTO DO SHOPPING HOSPEDEIRO.....18

3.2 APLICAÇÃO DO CINEMA.....26

3.3 APLICAÇÃO DA VÍDEO WEB LOCADORA.....30

4 CONSIDERAÇÕES FINAIS.....32

4.1 CONCLUSÕES.....32

4.2 TRABALHOS FUTUROS.....32

REFERÊNCIAS.....33

ÍNDICE DE FIGURAS

Figura 1: Arquitetura para Desenvolvimento de Aplicações Web adotada na disciplina de LPIII.....12

Figura 2: Exemplo de componente referenciando uma variável de um objeto no bean.....13

Figura 3: Entrar na Central de Rede e Compartilhamento do windows.....15

Figura 4: Habilitar compartilhamento de internet.....15

Figura 5: Habilitar duas opções do Microsoft .Net Framework 3.5.....16

Figura 6: Componentes adicionados a arquitetura base da aplicação.....18

Figura 7: Diretório raiz do servidor de aplicações com o arquivo de configuração do shopping hospedeiro.....19

Figura 8: Comando SQL para criação e exclusão de bases de dados.....19

Figura 9: Método isolado da classe GerenciadorDeDatasource responsável por enviar as requisições HTTP.....21

Figura 10: Interface FileStorage.....22

Figura 11: Página inicial do shopping.....23

Figura 12: Formulário de cadastro de aplicações.....23

Figura 13: Aplicação na listagem de lojas do shopping.....24

Figura 14: Botões para atualizar e remover a aplicação.....24

Figura 15: Janela de atualização de arquivo WAR.....25

Figura 16: Upload do arquivo WAR da aplicação.....25

Figura 17: Aplicação implantada pelo shopping hospedeiro.....26

Figura 18: Componentes adicionais usados para o desenvolvimento da aplicação do cinema. 27

Figura 19: Cinema dentro de uma tag iframe do lado esquerdo da página inicial do shopping. 28

Figura 20: Página de login da área administrativa do cinema……..28

Figura 21: Diagrama Entidade Relacionamento da aplicação do cinema.....29

Figura 22: Área administrativa do cinema.....29

Figura 23: Formulário de cadastro de filmes do cinema.....30

Figura 24: Página de detalhes do filme.....30

Figura 25: Cadastro de sessões de filme.....31

Figura 26: Diagrama ER da aplicação da vídeo locadora.....32

Figura 27: Tela de cadastro de diretores.....32

LISTA DE DEFINIÇÕES OU ACRÔNIMOS

API - Application Programming Interface

EJB - Enterprise JavaBeans

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IP - Internet Protocol

Java EE - Java Enterprise Edition

Java SE - Java Standard Edition

JDBC - Java Database Connectivity

JNDI - Java Naming and Directory Interface

JPA - Java Persistence API

JSF - Java Server Faces

JSON - JavaScript Object Notation

SQL - Structured Query Language

WAR - Web Application Archive

1 INTRODUÇÃO

O objetivo é o desenvolvimento de uma plataforma para auxiliar no ensino de desenvolvimento de aplicações web, que suporte serviços de hospedagem e persistência em banco de dados relacionais.

As instâncias dos processos do servidor de aplicação e banco de dados serão distribuídas nos computadores dos próprios alunos nas aulas práticas de laboratório. Os processos poderão se comunicar através de uma rede local criada com os notebooks presentes sem a necessidade de hardware adicional.

Três aplicações foram desenvolvidas no presente trabalho com os nomes temáticos de: Shopping Virtual, Cinema e Vídeo Web Locadora. Os temas são apenas um recurso didático e possuem significados em seu contexto.

A aplicação do Shopping é a plataforma de hospedagem. O shopping permite que desenvolvedores (alunos) enviem aplicações. Essas aplicações são as lojas do shopping. O shopping fornece um espaço no servidor de aplicações para que a aplicação possa ser executada e também uma bases de dados no sistema gerenciador de banco de dados.

As aplicações do Cinema e da Vídeo Web Locadora servirão como base para testes e modelo para desenvolvimento de outras aplicações.

1.1 HISTÓRICO E MOTIVAÇÃO

No curso de Sistemas de Informação, no contexto da disciplina Linguagem de Programação III, os alunos exercitam o desenvolvimento de aplicações web na linguagem Java, utilizando JSF para a implementação de páginas dinâmicas com persistência no banco através de um modelo objeto-relacional.

A utilização de um modelo objeto-relacional permite que todas as operações de leitura e escrita na base de dados seja realizada a partir dos objetos implementados na aplicação, de tal forma que os desenvolvedores não precisam se preocupar com as tabelas e registros suportados pelo gerenciador da base de dados relacional. A tradução do modelo de objetos para o modelo relacional

é realizada pelo framework de persistência, que no contexto desse trabalho foi utilizado uma implementação da especificação JPA (Java Persistence API).

A implementação de um sistema web distribuído é limitada a uma aplicação localhost em função de restrições frequentemente existentes em laboratórios dos cursos de graduação. Máquinas de redes distintas não podem se comunicar e as redes sem fio são instáveis devido a sobrecarga de acesso. Estas limitações impedem a utilização da rede do laboratório para implantar o servidor de aplicações e o servidor de bancos de dados em máquinas distintas, para acessá-los a partir de um navegador localizado em uma terceira máquina. Demonstrando, dessa maneira, a arquitetura de um sistema web em três camadas.

Uma possível solução para contornar essa limitação é a criação de uma rede local entre as máquinas presentes que seja independente da rede do laboratório sem o uso de hardware adicional além dos próprios notebooks dos alunos.

Adicionalmente, a utilização de aplicações web disponíveis no mundo real, tais como sistemas de automação bancária ou de compras de passagens aéreas, é realizada através do acesso de um site localizado em um servidor remoto, da empresa fornecedora do serviço, utilizando navegadores da web.

A alternativa de implantar uma aplicação distribuída em um site, com seu servidor de aplicações e de bases de dados distribuídos em máquinas distintas enriquece o ensino de desenvolvimento de aplicações web.

1.2 OPORTUNIDADES E RELEVÂNCIA

Como uma alternativa para solucionar o problema descrito na seção anterior, foi idealizada a metáfora do site de um shopping que oferece o serviço de hospedagem para as aplicações web que comercializam, na internet, os produtos e serviços de suas lojas.

As lojas do shopping são empresas que alugam os espaços comerciais do shopping e se beneficiam do serviço de hospedagem de suas aplicações web.

Os servidores de aplicações e o gerenciador de banco dados do site do shopping serão implantados em notebooks disponibilizados pelos alunos durante as demonstrações realizadas em

aulas da referida disciplina. As aplicações dos alunos serão implantadas no servidor de aplicações do sistema shopping hospedeiro e utilizarão bases de dados gerenciadas por um único servidor de banco de dados. Neste contexto, os alunos fazem o upload de suas aplicações no site do shopping hospedeiro e acessam o site para executar suas aplicações. A conexão entre os notebooks dos alunos é realizada através da configuração de uma rede local sem fio. Um notebook será responsável por disponibilizar a rede para que os outros possam se conectar.

A proposta deste trabalho é o desenvolvimento de uma plataforma de ensino de aplicações web que implementa a metáfora do site do shopping hospedeiro das aplicações web desenvolvidas por grupos de alunos da disciplina.

Os grupos de alunos terão um grande de leque de alternativas para implementar aplicações web para: lojas de departamentos, lojas de roupas, restaurantes, serviços de venda e troca de pneus, compra de pacotes de viagens aéreas e terrestres, etc.

1.3 OBJETIVOS DO TRABALHO

O objetivo geral deste trabalho é o desenvolvimento de uma plataforma para o ensino de aplicações web, desenvolvidas na linguagem Java.

Os objetivos específicos são:

a) o desenvolvimento de uma plataforma que suporte a metáfora de um site de shopping hospedeiro, com o upload de aplicações web;

b) a implantação do servidor de aplicações e do servidor de base de dados em máquinas distintas, utilizando notebooks disponibilizados por alunos durante as aulas da disciplina;

c) a alternativa de interligação dos notebooks através de uma rede local sem fio;

d) o desenvolvimento de uma aplicação web para ilustrar a sua utilização na referida plataforma.

1.4 METODOLOGIA ADOTADA

A metodologia adotada pode ser descrita em cinco etapas.

- Primeira etapa: Desenvolvimento do site do shopping hospedeiro, suportando a seleção de lojas do shopping para utilizar uma aplicação web que disponibiliza os produtos e serviços comercializados por essas loja, baseado na funcionalidade de upload de aplicações web dos sistemas associados às lojas do shopping.

- Segunda etapa: Desenvolvimento de uma aplicação web para cadastro e consulta da programação das salas do cinema do shopping, para enriquecer o visual do site do shopping hospedeiro, inspirado no site do shopping da cidade de Dourados.

- Terceira etapa: Desenvolvimento de uma aplicação parcial de uma vídeo locadora, para ilustrar a utilização de uma aplicação web no site do shopping hospedeiro.

- Quarta etapa: Implantação e teste da plataforma do shopping hospedeiro utilizando um notebook para hospedar o servidor de aplicações, um segundo notebook para hospedar o servidor de banco de dados e, pelo menos, um terceiro notebook com o navegador do usuário, interligados por uma rede local sem fio. Realização de testes de upload de aplicação web e de acesso aos links do site.

- Quinta etapa: Documentação da plataforma desenvolvida.

1.5 CONTEÚDO DO TRABALHO

No Capítulo 2 são abordados os conceitos e definições técnicas relacionadas com este trabalho. Serão explicados assuntos sobre as tecnologias utilizadas para implementação do sistema do lado do cliente e servidor, bem como a criação de uma rede local sem fio.

No Capítulo 3 são descritos os desenvolvimentos das três aplicações que compõem a plataforma proposta, baseada na metáfora do Shopping hospedeiro.

No Capítulo 4 são listadas as conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são discutidas a arquitetura, a rede sem fio e o servidor de aplicações utilizados nesta proposta. Os passos de configuração e utilização destas tecnologias são detalhados para que o presente trabalho possa ser reproduzido posteriormente.

2.1 ARQUITETURA DO SISTEMA

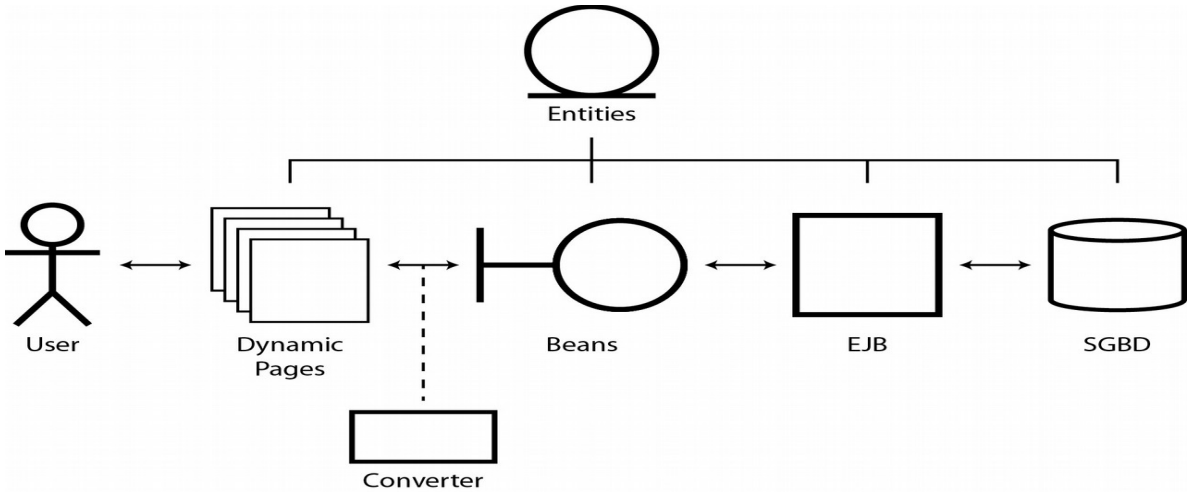

Figura i: Arquitetura para Desenvolvimento de Aplicações Web adotada na disciplina de LPIII. Fonte: SILVA, J. P. Desenvolvimento Web utilizando um Modelo Objeto-Relacional (2014).

A arquitetura, utilizada nesse trabalho, foi proposta em um Trabalho de Conclusão de Curso (SILVA, 2014). Essa arquitetura foi criada para atualizar a tecnologia do sistema desenvolvido no contexto do Projeto de Extensão “Fábrica de Software para Desenvolvimento de Sistemas de Governo Eletrônico” (BATISTA, 2014), e é utilizada em aulas da disciplina de Linguagem de Programação III.

JSF (Java Server Faces) foi utilizado como framework para criação de interfaces com o usuário. O JSF disponibiliza bibliotecas com uma grande variedade de componentes gráficos muito versáteis, que facilitam o desenvolvimento de interfaces para o usuário.

As interfaces de usuário se comunicam com os beans que, por sua vez, realizam chamadas aos objetos EJBs (Enterprise Java Beans). Os beans agem como intermediários entre a interface da aplicação e a lógica de negócios.

Os beans mantêm variáveis que refletem o estado da interface. As páginas da interface conseguem referenciar essas variáveis e popular os devidos campos nos beans. As páginas também conseguem chamar métodos dos beans para persistência ou busca de objetos. Nessa arquitetura isso é utilizado para fazer buscas dinâmicas, como por exemplo um campo de formulário que se auto-completa, ou seja, fornece uma sublista de strings que tem como prefixo um substring informado pelo usuário.

A Figura 2 ilustra a ligação entre a página dinâmica (arquivo xhtml) e o bean (arquivo java), a partir de um exemplo de componente de interface utilizado no shopping hospedeiro ligado a um objeto no bean. O objeto “cadastroDeAplicacaoBean”, injetado a partir do bean, contém a variável “aplicação” que representa o objeto que está sendo cadastrado pela interface. O valor preenchido em cada componente gráfico do cadastro é atribuído a um atributo do objeto, neste caso, o atributo “nome”. Após o preenchimento de todo o cadastro, o objeto estará completo, com todos os seus atributos informados pelo usuário, e poderá ser persistido na base de dados.

```
<p:inputText value="#{cadastroDeAplicacaoBean.aplicacao.nome}"
             placeholder="Digite o nome da aplicação" style="width: 280px;"
             required="true" requiredMessage="Digite o nome da aplicação"/>
```

Figura ii: Exemplo de componente referenciando uma variável de um objeto no bean

Os converters são necessários para traduzir objetos na forma de strings para a interface e posteriormente converter as strings novamente para objetos realizando o processo inverso. Eles são úteis para utilização em alguns componentes de interface.

Os EJBs são utilizados para encapsular a lógica de negócios e gerenciamento de transações. Toda consulta oriunda dos beans gerenciados do JSF passam pelo objetos EJBs que por sua vez realizam as operações de consulta e manipulação da base de dados utilizando a JPA (Java Persistence API).

Para fazer o mapeamento objeto relacional foi utilizada a especificação JPA como framework de persistência utilizando a implementação de referência EclipseLink. A JPA permite que seja feito um mapeamento em cada classe entidade das aplicações através de anotações java e

realiza a tradução do modelo de objetos para o modelo relacional suportando vários sistemas gerenciadores de banco de dados.

As entidades da aplicação percorrem todas as partes da arquitetura, sendo utilizadas desde a sua persistência no banco de dados, passando pelos EJBs e beans até a interface do usuário.

2.2 CRIAÇÃO DE REDE LOCAL SEM FIO

A alternativa escolhida para interligação das máquinas é a utilização de uma rede local sem fio adaptando qualquer notebook como um roteador. Como todo notebook possui uma placa de rede sem fio não será necessário utilizar hardware adicional.

O programa Virtual Wi-Fi Router[1] foi escolhido para suportar esta funcionalidade. No caso específico desse trabalho, este programa foi utilizado para criar um rede local entre os notebooks.

Ao utilizar esse programa é possível definir um nome e senha para a rede, e visualizar os endereços IPs dos usuários conectados. Essas informações serão compartilhadas com os alunos para que possam se conectar. Os endereços IPs são utilizados para configurar o shopping hospedeiro com o sistema gerenciador de banco de dados e para acessar o servidor de aplicações através das máquinas clientes.

Algumas configurações[2] devem ser feitas para o correto funcionamento do Virtual Wi-Fi Router no sistema operacional Microsoft Windows. Inicialmente, é preciso acessar “Central de Rede e Compartilhamento” (Figura 3), clicar em “Alterar Configurações do adaptador” encontrar a interface de rede desejada e clicar com o botão direito e escolher “Propriedades”. Acesse a aba “Compartilhamento” (Figura 4) e marque a opção “Permitir que outros usuários da rede se conectem pela conexão deste computador” e clique em “OK”.

Também é preciso realizar outra configuração. Para fazer isso é necessário entrar em “Painel de controle”, selecionar a opção “Programas” e clicar em “Ativar ou desativar recursos do Windows”, procurar pelo item “Microsoft .NET Framework” e ativar as duas opções dentro dele (Figura 5).

1 Virtual Wi-Fi Router (https://virtual-wi-fi-router.br.uptodown.com/windows)

2 Configurações Necessárias do Virtual Router http://www.baixaki.com.br/download/virtual-wi-fi-router.htm

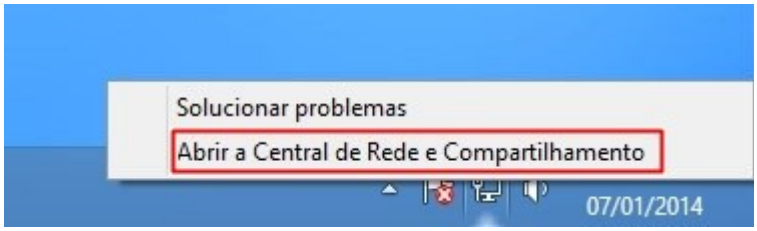

Figura 3: Entrar na Central de Rede e Compartilhamento do windows.

Fonte: http://www.baixaki.com.br/download/virtual-wi-fi-router.htm

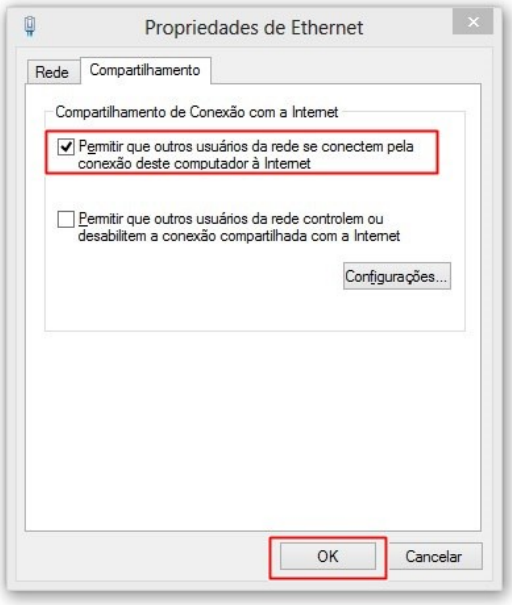

Figura 4: Habilitar compartilhamento de internet.

Fonte: http://www.baixaki.com.br/download/virtual-wi-fi-router.htm

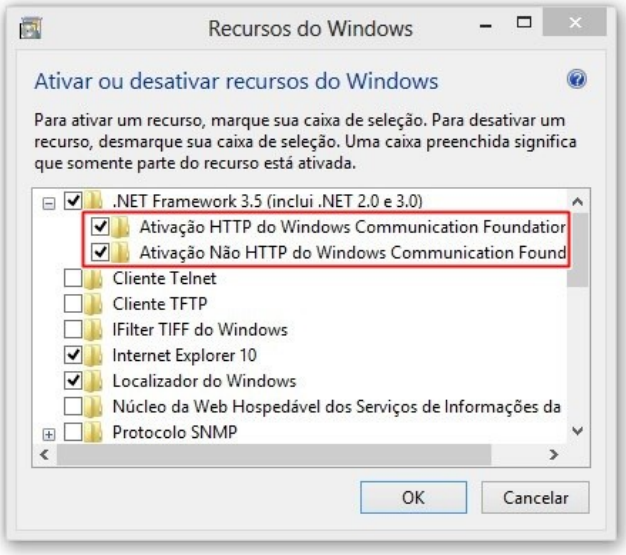

Figura 5: Habilitar duas opções do Microsoft .Net Framework 3.5. Fonte: http://www.baixaki.com.br/download/virtual-wi-fi-router.htm

2.3 SERVIDOR DE APLICAÇÕES WILDFLY

O servidor de aplicações WildFly foi utilizado para a implantação de um sistema web no contexto de um projeto de extensão (BATISTA, 2016), para o qual foi desenvolvido um tutorial de utilização da versão 8 do WildFly (HASEGAWA, 2014). Neste trabalho, está sendo utilizada a versão mais atual do WildFly (10).

O servidor de aplicações WildFly substituiu o GlassFish, que era utilizado anteriormente, por se mostrar mais robusto com base em testes realizados neste projeto de extensão. O WildFly 10 é uma rápida, leve e poderosa implementação das especificações Java EE 7 (JBOSS, 2016).

Na API JDBC bancos de dados são acessados usando datasources. Um objeto DataSource pode ser entendido como uma fábrica de conexões JDBC (Java Database Connectivity) que podem implementar pool de conexões de um uma fonte de dados em particular. Datasources podem ser registrados no serviço de diretórios do java (JNDI), de tal forma que aplicações podem usar a JNDI API para obter o objeto DataSource e se conectar a fonte de dados (ORACLE, 2016).

Datasources são configurados através de um subsistema do WildFly. Declarar um novo datasource consiste em fornecer um JDBC driver e definir uma nova configuração de datasource que referencia esse driver instalado (JBOSS, 2016).

Toda a configuração básica do WildFly é descrita no tutorial referido anteriormente (HASEGAWA, 2014). Após iniciar o WildFly basta acessar no browser o endereço do servidor na porta 8080, que é a porta http padrão. O WildFly pode ser administrado através de uma interface de linha de comando ou um console web. Para acessar o console web basta apontar o browser no endereço do servidor na porta 9990, que é a porta padrão do console de administração (JBOSS, 2016).

Adicionalmente para configurar a plataforma de ensino de aplicações web é preciso criar um datasource para o cinema e outro para o shopping hospedeiro.

3 DESENVOLVIMENTO DO TRABALHO PROPOSTO

Nesse capítulo são descritos detalhes da implementação das aplicações desenvolvidas bem como o seu funcionamento. Para os dois principais sistemas, shopping hospedeiro e cinema, são apresentados cenários de utilização das funcionalidades, associando com as funcionalidades providas pelas classes de objetos das aplicações, para facilitar o entendimento e servir de base para a evolução deste trabalho. As três aplicações, desenvolvidas neste trabalho, são projetos independentes. O cinema é visualizado como uma seção interna à página do shopping. A aplicação da vídeo locadora serve como base para testes de upload de aplicações no shopping hospedeiro.

3.1 DESENVOLVIMENTO E EXPLICAÇÃO DO FUNCIONAMENTO DO SHOPPING HOSPEDEIRO

O desenvolvimento do shopping hospedeiro segue o modelo de arquitetura para desenvolvimento de aplicações na disciplina de Linguagem de Programação III, com a adição de componentes para realizar algumas tarefas referentes a toda atividade de uploads de aplicações. Veja na Figura 6 os pacotes adicionados.

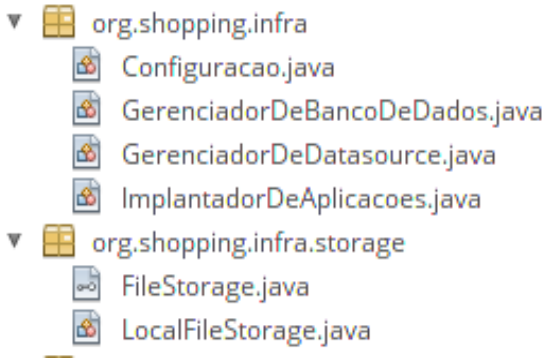

Figura 6: Componentes adicionados a arquitetura base da aplicação.

Primeiramente será explicado qual o papel de cada classe e em seguida o fluxo para o cadastro de uploads de aplicações.

A classe Configuracao é responsável por obter valores de algumas configurações em um arquivo localizado na pasta raiz do servidor de aplicações que está executando o shopping hospedeiro. O nome do arquivo deve ser exatamente “configuracao-shopping.properties” (Figura 7). A leitura do arquivo é realizado pela classe de propriedades Properties da API base do Java SE. Esse arquivo de configuração é útil para guardar valores que vão mudar constantemente, como por exemplo, o endereço do host que é responsável por executar o banco de dados.

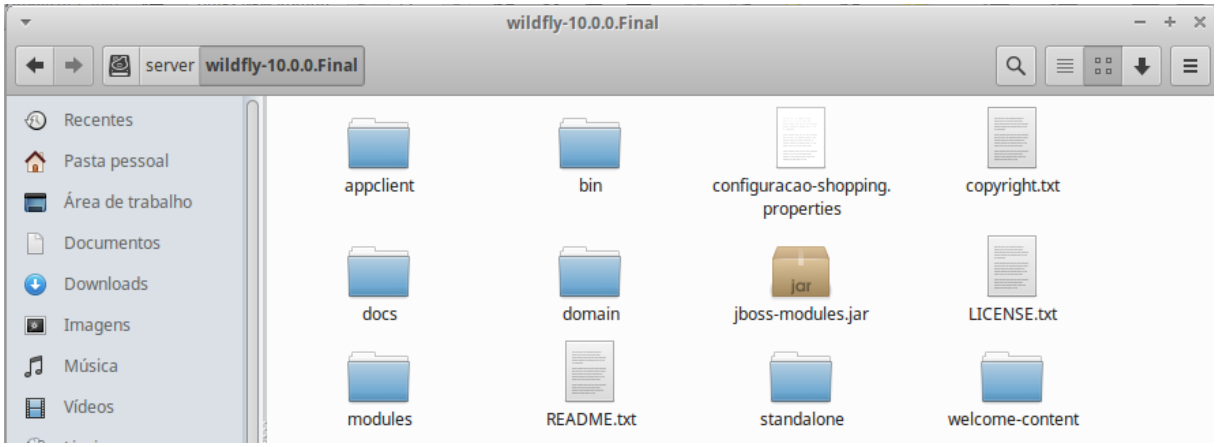

Figura 7: Diretório raiz do servidor de aplicações com o arquivo de configuração do shopping hospedeiro.

A classe GerenciadorDeBancoDeDados é responsável por executar os comandos SQL de criação e exclusão das bases de dados utilizadas pelas aplicações enviadas para o shopping. Os comandos SQL utilizados são demostrados na Figura 8.

```
Comando SQL para criação das bases de dados:
    CREATE DATABASE IF NOT EXISTS nome_da_base_de_dados CHARACTER SET utf8 COLLATE utf8_general_ci


Comando SQL para remover as bases de dados:
    DROP DATABASE IF EXISTS nome_da_base_de_dados
```

Figura 8: Comando SQL para criação e exclusão de bases de dados.

A classe GerenciadorDeDatasource tem a função de criar datasources no WildFly. O método utilizado para fazer isso se dá por meio do envio de requisições HTTP (Hypertext Transfer Protocol) do tipo POST com conteúdo no formato JSON (JavaScript Object Notation)[3] para API de gerenciamento do WildFly.

3 JSON http://www.json.org/

A biblioteca utilizada para enviar as requisições HTTP é a Apache HttpComponents[4]. Apesar de ser uma biblioteca complicada de se utilizar no que diz respeito a complexidade da API, ela foi escolhida por suportar o método de autenticação Http Digest exigido pelo WildFly. Uma outra alternativa que suporta uma utilização mais simples seria a biblioteca Unirest[5], que geraria um código menos verboso, mas essa biblioteca não suporta o método de autenticação necessário.

O código utilizado para realizar a requisição HTTP foi adaptado de trechos de outros códigos presentes no manual da biblioteca utilizada. A Figura 9 exibe o método java responsável por enviar as requisições HTTP. O método recebe como único parâmetro um conteúdo em string no formato JSON que será anexado no corpo da requisição HTTP.

O comando no formato JSON enviado para criar o datasource possui a seguinte estrutura específica e é definida pelo WildFly:

```
{
        "operation":"add",
        "address":[{"subsystem": "datasources"}, {"data-source": "NomeDoDatasource"}],
        "connection-url": "jdbc:mysql://localhost:3306/nome_do_banco_de_dados",
        "driver-name": "nome_do_driver_mysql_configurado_no_wildfly"
        "jndi-name": "java:jboss/datasources/NomeDoDatasource"
        "user-name": "usuario_mysql"
        "password": "senha_mysql"
}
```

O formato JSON possui um estrutura de chaves e valores. A primeira chave “operation” contém o valor da operação no Wildfly. A segunda chave “address” contém uma array de objetos JSON que significam o endereço do recurso no Wildfly que nesse caso é o endereço do datasource. A terceira chave “connection-url” contém a url de conexão com o banco de dados. A quarta chave “driver-name” contém o nome do driver do banco de dados configurado no Wildfly. A quinta chave “jndi-name” é o endereço do serviço de diretórios do java onde o datasource vai estar disponível para ser posteriormente obtido e utilizado pelas aplicações.

4 Apache HttpComponents https://hc.apache.org/

5 Unirest http://unirest.io/java.html

```java
private boolean enviarRequisicaoHttp(String conteudoDaRequisicaoJson) {

        // objeto HttpHost com o endereço e porta da área de administração do WildFly
        HttpHost httpHost = new HttpHost(hostAdminWildfly, portaAdminWildfly, "http");
        HttpClientContext context = HttpClientContext.create();

        // objeto HttpPost com o caminho "/management" da API de gerenciamento do WildFly
        HttpPost httpPost = new HttpPost("/management");

        // o conteúdo da requisição http é definido como sendo em JSON
        httpPost.setHeader("Content-type", "application/json");

        try {
            // defino a mensagem em json como corpo da requisição
            httpPost.setEntity(new StringEntity(conteudoDaRequisicaoJson));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            return false;
        }

        // configuração do método de autenticação
        CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials
        context.setCredentialsProvider(credentialsProvider);

        HttpClient client = HttpClients.createDefault();
        HttpResponse response;
        try {
            // dispara a requisição HTTP
            response = client.execute(httpHost, httpPost, context);
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }

        return response.getStatusLine().getStatusCode() == 200;
}
```

Figura 9: Método isolado da classe GerenciadorDeDatasource responsável por enviar as requisições HTTP.

O comando em formato JSON enviado para excluir um datasource no Wildfly possui o seguinte formato:

```
{
        "operation": "remove",
        "address": [{"subsystem": "datasources"}, {"data-source": "NomeDoDatasource"}]
}
```

A classe LocaFileStorage tem a função de salvar, remover e obter os arquivos que foram enviados. Os arquivos são armazenados no sistema de arquivos local onde o servidor de aplicações está executando. Foi utilizada a biblioteca Apache Commons IO[6] para auxiliar no desenvolvimento das operações de entrada e saída referentes as manipulações de arquivos.

A classe LocalFileStorage é uma implementação da interface FileStorage. Caso seja necessário armazenar os arquivos em outro lugar, basta criar uma nova implementação da interface

6 Apache Commons IO https://commons.apache.org/proper/commons-io/

FileStorage e atualizar a aplicação para utilizá-la. Na Figura 10 é ilustrada a interface FileStorage com os métodos que necessitam ser implementados. É sempre uma boa prática programar com foco na interface e não na implementação. (SILVEIRA, 2012)

```
1  package org.shopping.infra.storage;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.io.InputStream;
6
7  public interface FileStorage {
8
9      public File salvarArquivo(InputStream inputStream, String pasta, String nomeDoArquivo) throws IOException;
10
11     public void removerPasta(String pasta) throws IOException;
12
13     public boolean arquivoExiste(String pasta, String nomeDoArquivo);
14
15     public File obterArquivo(String pasta, String nomeDoArquivo);
16
17     public void removerArquivo(String pasta, String nomeDoArquivo);
18 }
19
```

Figura 10: Interface FileStorage.

A classe ImplantadorDeAplicacoes tem a função de implantar e remover a implantação das aplicações enviadas para o shopping hospedeiro. O método utilizado para realizar isso é movendo e removendo os arquivos WAR (Web Application Archive) para a pasta de deployments do WildFly.

O envio de aplicações no shopping hospedeiro consiste de duas etapas. Primeiro é necessário fazer o cadastro e após isso será possível fazer o upload da aplicação quantas vezes for necessário.

O cadastro pode ser feito acessando o formulário ao clicar no link “Cadastrar aplicação” na página inicial do shopping (Figura 11).

No formulário de cadastro de aplicações (Figura 12) é necessário preencher os campos com o nome da aplicação, o contexto em que vai estar disponível, o nome do datasource, escolher um segmento para a loja, uma senha e escolher uma imagem para exibição na página inicial.

O contexto e o datasource de cada aplicação são únicos. O contexto é o endereço base em que a aplicação vai estar disponível. O nome escolhido para o datasource será concatenado com o prefixo “java:jboss/datasources/” pelo sistema para formar o endereço JNDI. A senha fornecida será utilizada posteriormente para que o usuário possa se autenticar para atualizar o arquivo WAR e remover a aplicação. O administrador do shopping possui uma senha que é válida para realizar essas operações em todas as aplicações. Essa senha é definida no arquivo de configuração do shopping com o nome de propriedade ‘senha.administrador’.

Após o cadastro a aplicação aparecerá na listagem de lojas da página inicial do shopping (Figura 13). É possível realizar uma busca direta por lojas pelo título e filtrar por segmento.

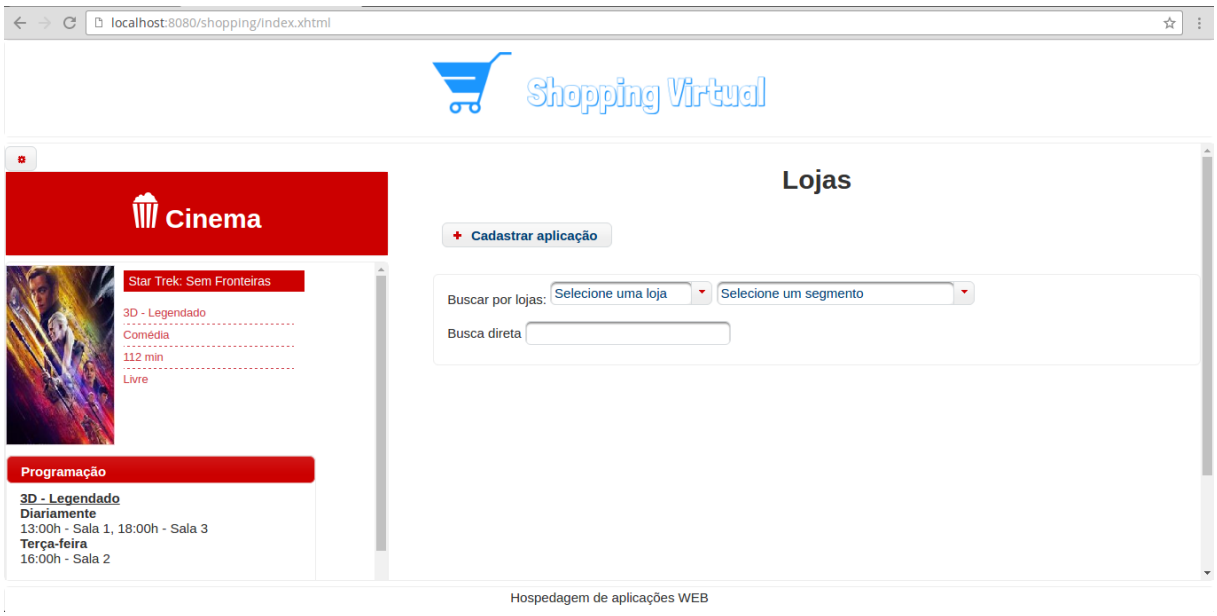

Figura 11: Página inicial do shopping.

Figura 12: Formulário de cadastro de aplicações.

Figura 13: Aplicação na listagem de lojas do shopping.

A partir desse momento será possível fazer o upload do arquivo WAR da aplicação clicando no botão de atualização (Figura 14).

Figura 14: Botões para atualizar e remover a aplicação.

Ao clicar no botão de atualização será aberta uma janela pedindo a senha que foi fornecida no cadastro da aplicação (Figura 15).

Figura 15: Janela de atualização de arquivo WAR.

Após se autenticar, basta escolher e enviar o arquivo WAR da aplicação (Figura 16). Nesse momento, o shopping hospedeiro irá criar uma base de dados no sistema gerenciador de banco de dados, criar um datasource no WildFly e implantar a aplicação. Após isso ela já estará em execução e será possível acessá-la (Figura 17).

Figura 16: Upload do arquivo WAR da aplicação.

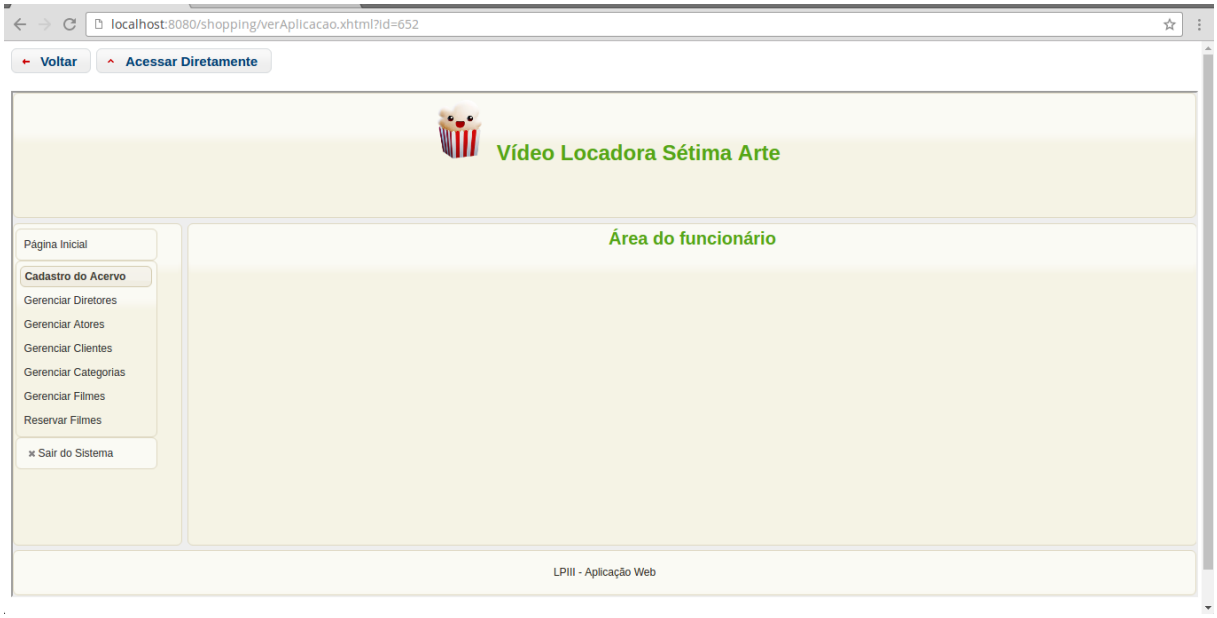

Figura 17: Aplicação implantada pelo shopping hospedeiro.

3.2 APLICAÇÃO DO CINEMA

O cinema está incorporado em uma seção do lado esquerdo da página inicial do shopping dentro de uma tag iframe do HTML (Figura 19). A tag iframe permite embutir uma página web dentro de outra.

A aplicação do cinema suporta o cadastro de filmes e sessões para os filmes. Cada sessão de filme é composta por um tipo de vídeo e áudio e possui várias configurações diárias que são as configurações dos dias da semana. Cada configuração diária possui várias configurações de horários. A Figura 21 exibe o diagrama entidade relacionamento da aplicação do cinema.

Foi necessário adicionar alguns componentes à arquitetura de aplicações da disciplina de Linguagem de Programação III, para desenvolvimento do Cinema (Figura 18). Todos eles são relativos a manipulações das imagens dos filmes. A classe Imagem é uma entidade utilizada internamente pela aplicação mas que não necessita ser persistida no banco de dados. A classe ImagensServlet é uma servlet para servir imagens, tendo em vista que essas imagens não estão em um diretório público no servidor web. A classe LimpezaDeArquivosTemporarios serve para limpar o diretório onde os arquivos de imagens são armazenados temporariamente enquanto o usuário preenche o formulário de cadastro de filmes. As classes RepositorioDeImagensDeFilmes e

RepositorioDeImagensTemporarias estendem a classe abstrata RepositorioDeImagens e tem a função de salvar, remover e obter imagens de filmes e temporárias respectivamente.

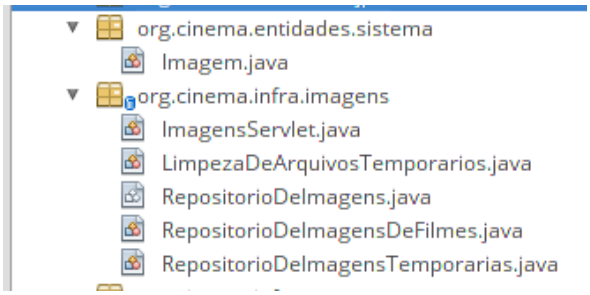

Figura 18: Componentes adicionais usados para o desenvolvimento da aplicação do cinema.

No canto superior esquerdo da seção do cinema existe um botão que permite acessar a área administrativa do cinema (Figura 19). Para acessar a área administrativa é necessário inserir a senha do administrador do cinema (Figura 20). A senha do administrador do cinema não é a mesma do shopping e é definida no código fonte no bean ‘AutenticacaoBean’ na váriavel ‘senhaDoAdministrador’ do método ‘autenticar’. A página inicial da área administrativa exibe uma listagem de todos os filmes e permite cadastrar novos (Figura 22). Após o cadastro é possível acessar a página de detalhes do filme para cadastrar as sessões (Figura 24).

O cadastro de sessões (Figura 25) funciona da seguinte maneira: adicionam-se os horários das salas; em seguida adiciona-se a configuração diária, repetindo-se esses dois passos quantas vezes forem necessários; e por último basta selecionar o tipo de vídeo e áudio e salvar a sessão de filme.

Figura 19: Cinema dentro de uma tag iframe do lado esquerdo da página inicial do shopping.

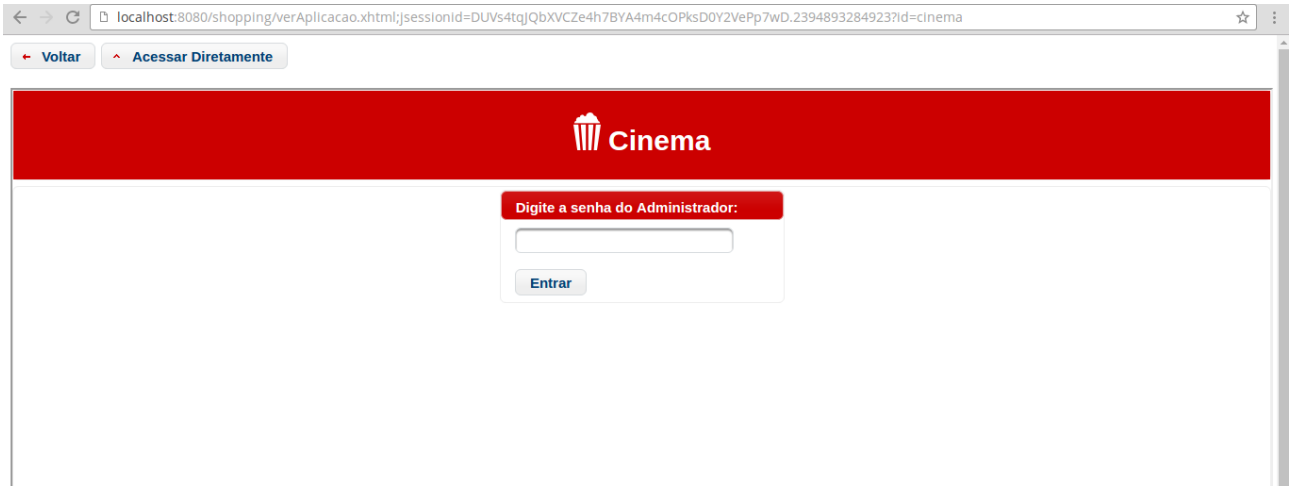

Figura 20: Página de login da área administrativa do cinema.

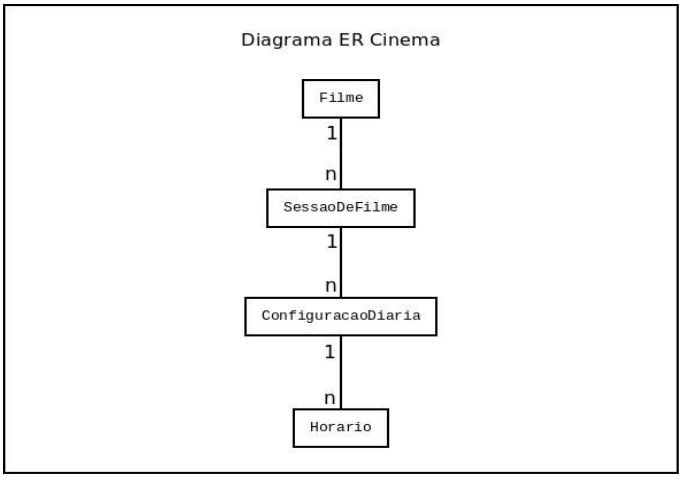

Figura 21: Diagrama Entidade Relacionamento da aplicação do cinema.

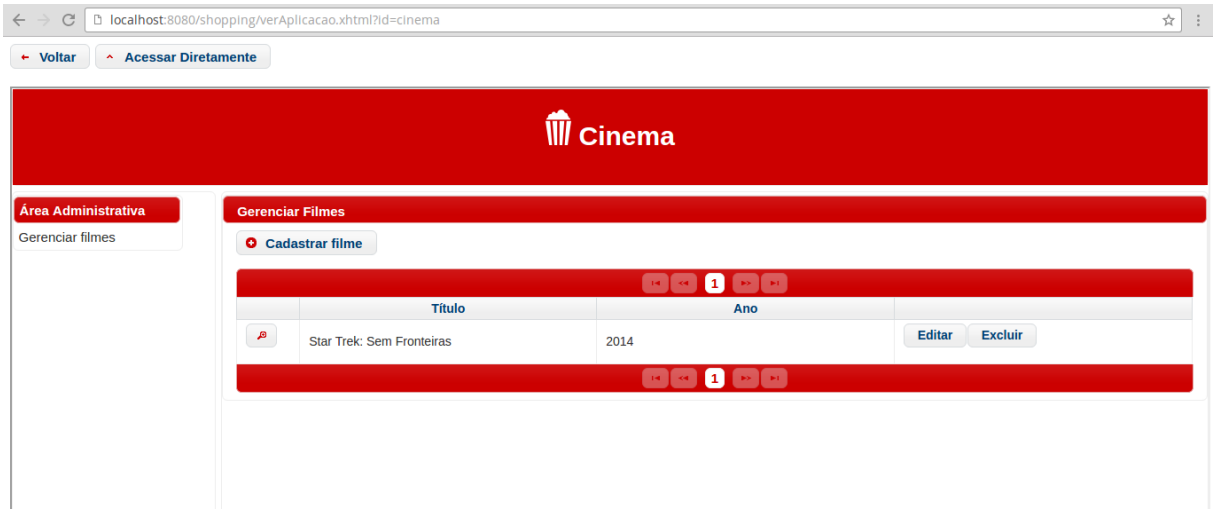

Figura 22: Área administrativa do cinema.

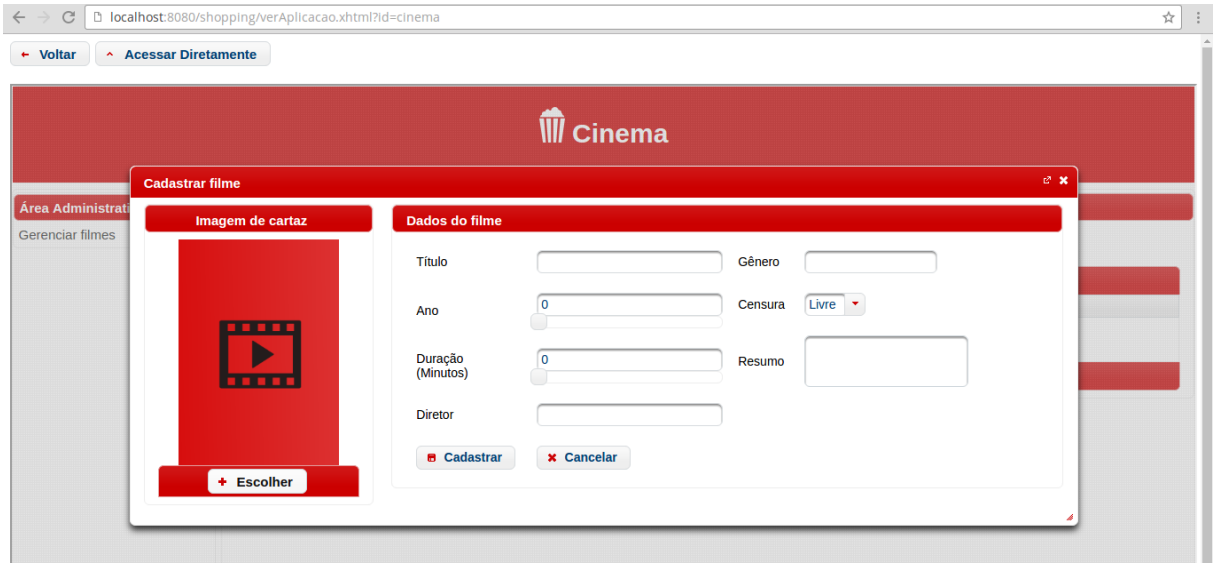

Figura 23: Formulário de cadastro de filmes do cinema

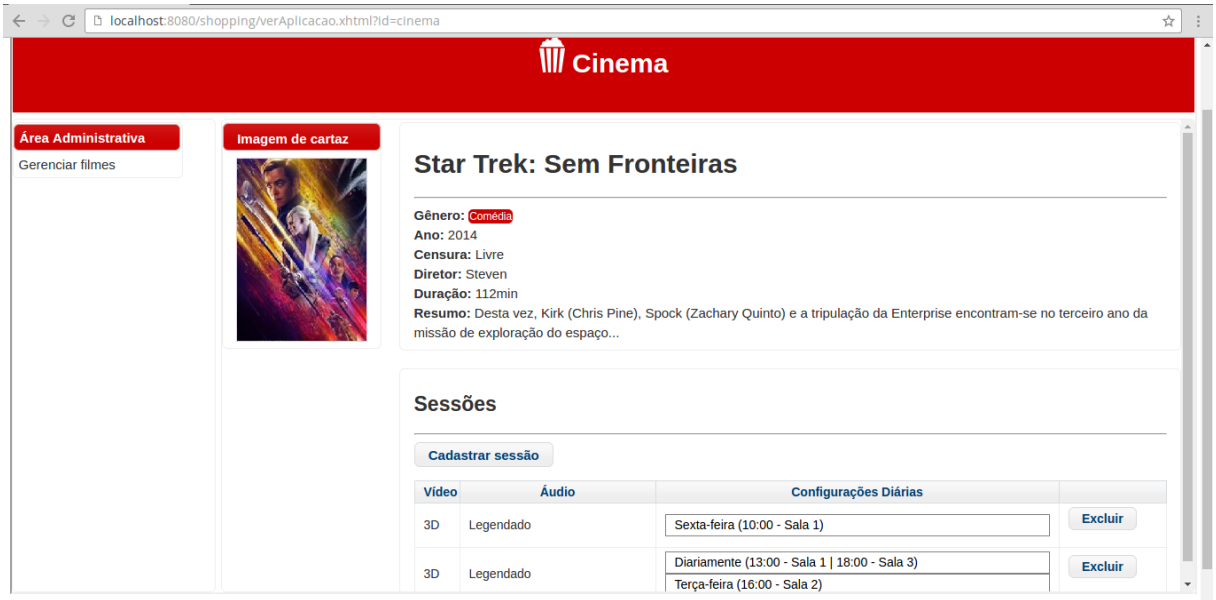

Figura 24: Página de detalhes do filme

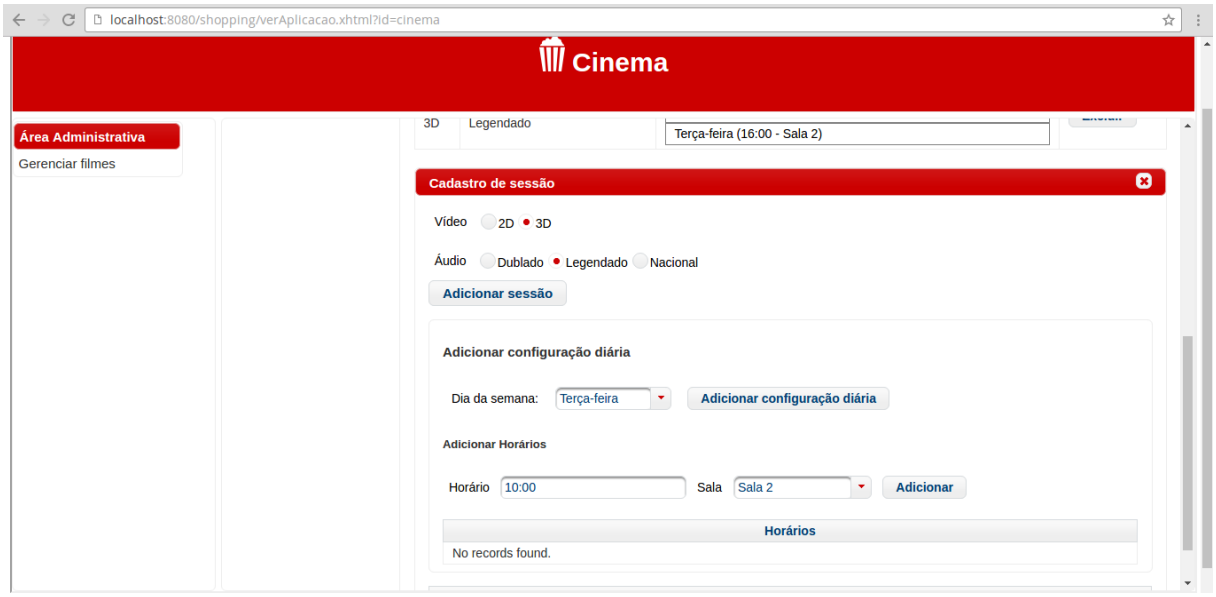

Figura 25: Cadastro de sessões de filme.

3.3 APLICAÇÃO DA VÍDEO WEB LOCADORA

A aplicação da Vídeo Web Locadora foi desenvolvida anteriormente em projetos de extensão e serve como exemplo e base para testes. O desenvolvimento foi realizado seguindo os execícios de um tutorial da disciplina de Linguagem de Programação III. A implementação segue a arquitetura dessa disciplina.

A Vídeo Web Locadora permite cadastrar diretores, atores, clientes, categorias e filmes e realizar reservas dos filmes para os clientes. A Figura 26 exibe o diagrama entidade relacionamento dessa aplicação e a Figura 27 exibe um exemplo da tela de cadastro de diretor.

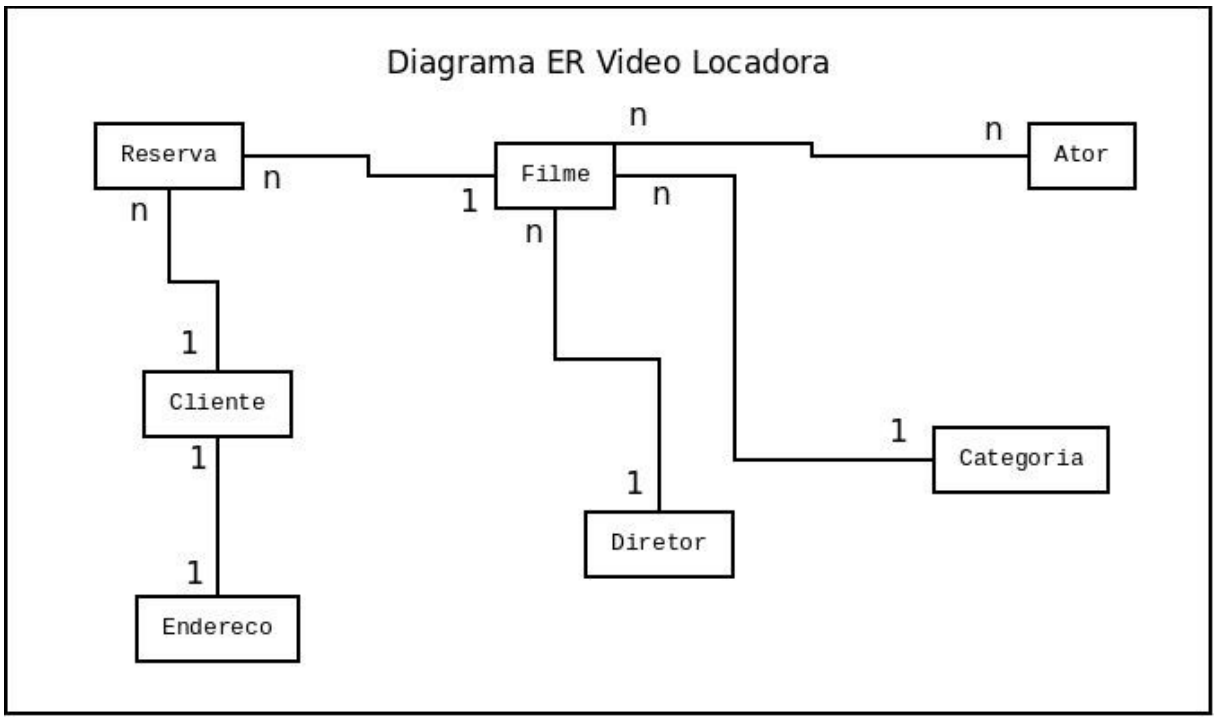

Figura 26: Diagrama ER da aplicação da vídeo locadora

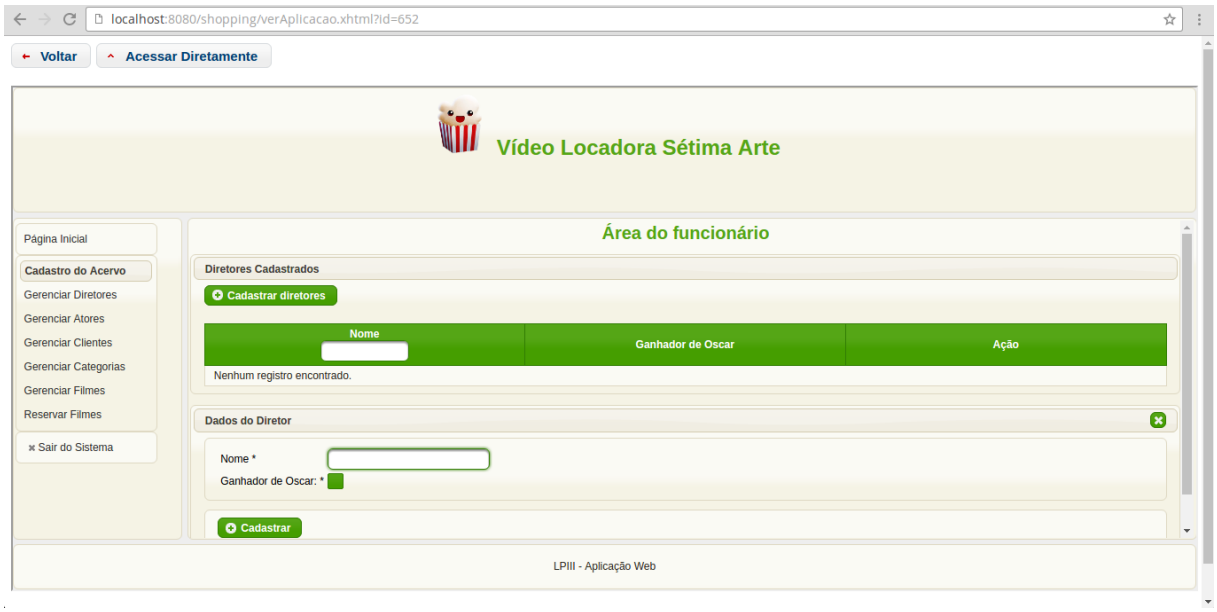

Figura 27: Tela de cadastro de diretores.

4 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões e sugestões para trabalhos futuros.

4.1 CONCLUSÕES

A interface, provida pela plataforma desenvolvida neste trabalho, é muito versátil para que alunos possam realizar o upload de aplicações desenvolvidas, no contexto da disciplina Linguagem de Programação III. A metáfora do shopping hospedeiro de aplicações de associadas à lojas do shopping é muito útil para que os alunos desenvolvam aplicações web, hospedem suas aplicações em um site e visualizem a utilização de uma aplicação web em camadas distintas na web com clara visão da separação em camadas de apresentação, lógica de negócios e persistência dos dados da aplicação.

Trabalhos como este abrem oportunidades de melhor utilização das tecnologias disponíveis que fazem parte do nosso cotidiano, como por exemplo, a criação de uma rede local WiFi com a utilização dos computadores dos próprios alunos.

4.2 TRABALHOS FUTUROS

Seria desejável evoluir a plataforma proposta neste trabalho em dois aspectos: melhor separação do ponto de vista da interface, de funcionalidades administrativas do site (cadastro do cinema e implantação de aplicações), das funcionalidades dos usuários (visualização da programação do cinema e utilização dos sistemas associadas às lojas do shopping); evolução do suporte à autenticação e autorização de recursos para os usuários.

REFERÊNCIAS

SILVEIRA, P.; et al. **Introdução à arquitetura e design de software – Uma visão sobre a plataforma java.** Rio de Janeiro: Elsevier, 2012.

SOUZA, A. **Java EE: aproveite toda a plataforma para construir aplicações.** Casa do Código, 2015.

MOZILLA. Mozilla Developer Network. **<iframe>.** Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe>. Acesso em: 6 de Julho de 2014.

JBOSS. **Getting Started with WildFly 10.** Disponível em: https://docs.jboss.org/author/display/WFLY10/Getting+Started+Guide. Acesso em 18 de Setembro de 2016.

ORACLE. **The Java EE 5 Tutorial.** Disponível em: http://docs.oracle.com/javaee/5/tutorial/doc/bncjj.html. Acesso em 18 de Setembro de 2016.

SILVA, João Paulo. **Desenvolvimento Web utilizando um Modelo Objeto-Relacional.** Trabalho de Conclusão do Curso de Sistemas de Informação, FACET/UDGD, 2014.

BATISTA Jr, Joinvile. **Fábrica de Software para Desenvolvimento de Sistemas de Governo Eletrônico.** Projeto de Extensão, FACET/UFGD, 2010 a 2014.

BATISTA Jr, Joinvile. **Implantação e Manutenção Corretiva e Evolutiva do Sistema Web Agência Municipal de Empregos.** Projeto de Extensão, FACET/UFGD, 2015 a 2016.

HASEGAWA, Marcelo Yoshio. **Configurações do WildFly 8.0.0.** Anexo do Projeto de Extensão “Implantação e Manutenção Corretiva e Evolutiva do Sistema Web Agência Municipal de Emprego”. 2014.