Implementação de um Web Service REST Aplicando Injeção de Dependências e Contextos (CDI)

Luiz Felipe Fronchetti Dias[1], João Martins de Queiroz Filho[1], Vinícius Ribeiro Morais[1]
[1]Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87.301-899 – Paraná – PR – Brasil

São diversas as soluções possíveis para integração de sistemas e comunicação entre aplicações diferentes. Entre estas soluções, se encontra o conceito de *Web Service*, muito útil para grandes aplicações, visto que gerencia dados através da própria Web. Este artigo descreve a implementação de um *Web Service REST* feito na linguagem de programação Java, onde foi aplicada a especificação do Java EE 6 denominada Injeção de Dependências e Contextos (CDI), responsável pela inversão de controle entre classes. Os resultados obtidos ao fim desta implementação foram quatro: um serviço de administração de dados para uma empresa, duas aplicações para clientes e um serviço web.

***Index Terms*—sistemas distribuídos, webservice, cdi, injecao de dependencia, rest, json, jarsey.**

I. Introdução

O Trabalho proposto para a disciplina de Sistemas Distribuídos foi implementar um *Web Service* aplicando CDI(*Context Dependency Injection*). Assim, foi decidido que o projeto seria formado por quatro elementos, sendo eles:*Web Service* com CDI, interface para o administrador do *Web Service*, um *website* para expor as informações e um aplicativo. Então, a ideia foi criar um sistema de Cinema, ou seja, um sistema que o dono possa expor os filmes de seu cinema tanto no *website* quanto em um aplicativo.

A estrutura do artigo está organizada da seguinte forma: na sessão II será apresentado a fundamentação teórica mostrando os conceitos dos principais *Web Services* utilizados ultimamente e, na sessão III, será mostrado todas as ferramentas que foram utilizadas durante a implementação. A sessão IV traz uma breve descrição da arquitetura utilizada no sistema e como foi feita a implementação da interface para o cliente e o administrador. A sessão V mostra os resultados obtidos no objeto e algumas ilustrações das interfaces. Por fim, a sessão VI traz a conclusão.

II. Fundamentação Teórica

Segundo a organização W3C (*World Wide Web Consortium*), um *Web Service*[2] é um sistema de software responsável por proporcionar a interação entre duas máquinas. Esta interação ocorre através de uma interface descrita em um formato específico, e permite que sistemas interajam com um *Web Service* usando esta interface. A comunicação entre sistemas e interface ocorre partir da troca de mensagens, seguindo protocolos definidos para este tipo de serviço. Dentre os protocolos mais usados pela comunidade se encontram o protocolo *SOAP* e o *REST*. As mensagens *SOAP* basicamente são documentos *XML* (*eXtensible Markup Language*) serializados segundo o padrão *W3C*, que são enviados em cima de um protocolo de rede (Por exemplo, *HTTP*). O problema deste padrão, é que ele adiciona um *overhead* considerável, tanto por ser em *XML* quanto por adicionar muitas tags de meta-informação. Visto tais desvantagens e o escopo do projeto em questão, para implementação deste *Web Service* um protocolo mais flexível foi utilizado, o *REST*.

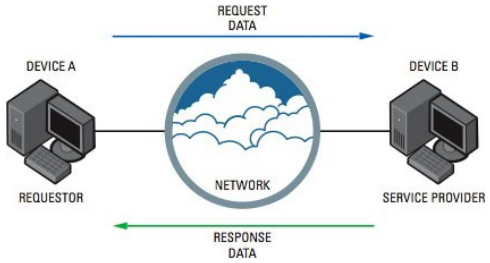

Figura 1. Exemplo de comunicação: Web Service e Sistemas

A. REST

Representational State Transfer (*REST*), em português Transferência de Estado Representacional, é outro protocolo de comunicação entre sistemas e interface, implementado sobre o protocolo *HTTP* (*Hypertext Transfer Protocol*) mencionado anteriormente. Ser implementado sobre o protocolo *HTTP*, quer dizer que todos os métodos deste podem ser utilizados pelo *REST*. A maior vantagem deste protocolo é a sua flexibilidade. O desenvolvedor pode optar pelo formato mais adequado para responder mensagens aos sistemas, de acordo com sua necessidade. Os formatos mais comuns nesta comunicação são o *JSON* (*JavaScript Object Notation*)[1], *HTML* (*HyperText Markup Language*) e o próprio *XML* usado no *SOAP*[3]. No *Web Service* implementado neste projeto, a comunicação acontece através do formato *JSON*, escolhido por razões de legibilidade, velocidade de execução e tamanho de arquivo.

III. Materias e Métodos

A. Jersey: Framework para Web Services REST

Uma aplicação para ser considerada do tipo *REST*, deve seguir todos os padrões desta arquitetura, utilizando de operações tais como *GET*, *PUT*, *DELETE* e *POST* durante toda

sua concepção. Entretanto, construir uma aplicação *REST* do zero requer um trabalho árduo e desnecessário, sendo que existem diversos *frameworks* que implementam grande parte do serviço. Um dos principais *frameworks* para desenvolver aplicações do tipo *REST* em Java é o *Jersey*.

Desenvolvido pela *GlassFish*[1], o *Jersey* provê uma biblioteca de implementação de *Web Services REST* através de um *Java Servlet Container*. O *servlet* analisa as requisições *HTTP* feitas pelos clientes, e seleciona a classe e método correto para responder a cada requisição. Esta seleção é possível graças as *annotations*[2] feitas nas classes e métodos do *Web Service*. Logo, para acessar um determinado recurso do *Web Service*, requisitamos na *url* o caminho de uma classe ou método e recebemos o recurso como resposta. Confira abaixo um exemplo de classe similar as implementadas neste *Web Service*:

```
import javax.ws.rs.*;

/* Especifica o caminho ate a classe */
@Path("/hello")
public class Pagina {
/* Especifica o caminho ate a metodo */
@Path("/world")
/* Especifica o metodo http usado */
@GET
/* O que ele recebe */
@Consumes(MediaType.TEXT_PLAIN)
/* O que ele responde */
@Produces(MediaType.APPLICATION_JSON)
public String HelloWorld(String message){
return "Hello World!" + message;
}
}
```

```
Neste exemplo, o caminho usado pelo cliente
seria:
http://dominio:porta/meu_projeto/hello/world
```

Durante a implementação, visando trabalhar em cima do contexto proposto, um pacote "Páginas"foi criado para agrupar todas as classes usadas pelo *Jersey* neste projeto. O pacote agrupou quatro categorias divididas em classes distintas: Comentários, Filmes, Usuários e Página Inicial, cada qual relacionada a sua respectiva função no *Web Service*.

(a) **Comentários:** Agrupa todos os métodos relacionados aos comentários feitos pelos usuários nos filmes.
(b) **Filmes:** Agrupa todos os métodos relacionados aos filmes cadastrados e suas avaliações.
(c) **Usuários:** Agrupa todos os métodos relacionados aos usuários cadastrados, sistema de cadastro e *login*.
(d) **Página Inicial:** Página de início apresentada pelo *Web Service*

Como requisitado pelo *framework*, este pacote foi referenciado em um arquivo de marcação denominado *web.xml*, usado pelo *Jersey* para encontrar as classes relacionadas ao *servlet*. Um pacote de interfaces para especificação dos métodos usados e um pacote para códigos relacionados ao banco de dados também foram criados, entretanto, ambos não necessitaram constar neste arquivo de marcações, visto que fazem parte do "*backend*"da aplicação.

[1]https://glassfish.java.net/

[2] *Conjunto de meta-dados ao longo do código que podem ser posteriormente interpretados por um compilador*

B. Apache Tomcat: Servidor Web Java

Para rodar uma aplicação *web*, é necessário um servidor que gerencie as requisições e faça uma ponte entre a aplicação implementada e os clientes. O *Web Service* por si só é apenas uma aplicação, e também necessita de um servidor para hospedagem. Como a linguagem escolhida foi Java, se deu por necessário o uso de um servidor *web* com suporte para a linguagem. O *Tomcat*, desenvolvido pela Fundação *Apache* [3], foi o servidor escolhido, pois é centrado na linguagem de programação Java, mais especificamente nas tecnologias de *Servlets* e de *Java Server Pages (JSP)*. A sua configuração é extremamente simples, visto que o serviço passa a funcionar logo após sua instalação. A versão escolhida pelos desenvolvedores foi a *Tomcat* 7, visto que este não é só atual como também estável se comparado ao *Tomcat* 8.

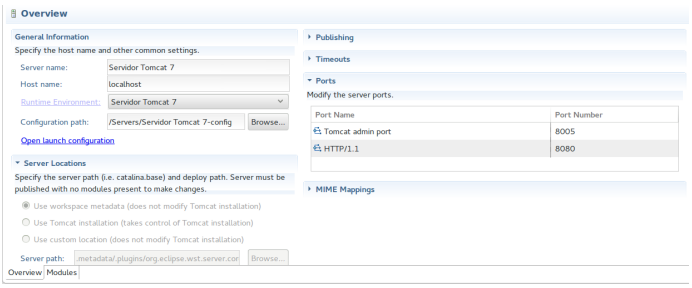

Figura 2. Página de configuração do *Tomcat* 7 (Eclipse)

C. Inversão de Controle: Injeção de Dependências

A Inversão de Controle é uma forma diferente, definida por um padrão de projeto, que temos para manipular o controle sobre um objeto. A inversão de controle, de modo superficial, pode ser definida como sendo a mudança de conhecimento que uma classe tem em relação à outra. É comum que em classes complexas como as de um *Web Service*, uma classe acabe dependendo de outras classes. Por exemplo, uma classe Pagina dependa de classes tais como *JSONObject* e *HttpURLConnection*. Isto, de um ponto de vista arquitetural, pode acabar sendo não só um falha de desenvolvimento, como uma tremenda dor de cabeça, visto que para quaisquer mudanças em um determinado *JSONObject*, será necessário acessar a classe Pagina.

São diversas as maneiras para se executar uma inversão de controles, uma delas é a injeção. A injeção de dependências aplica a classe que depende à classe que a instancia, desta forma, ao invés de deixarmos a responsabilidade da criação da classe *JSONObject* para a classe Pagina, vamos dar a Pagina esta dependência. Em um exemplo prático e similar aos aplicados no projeto, temos:

[3]https://www.apache.org/

```
public class Pagina {
        private JSONObject j;
        /* A declaracao de "j" faz com que */
        /* haja inversao de controle */
 public void defineMessage(JSONObject message){
        this.j = logVenda;
 }
 public void returnMessage() {
        return j;
 }
}
```

As injeções de dependências foram aplicadas por todo o projeto, principalmente em classes relacionadas ao banco de dados, logo que nestas a geração de dependências é grande e pode se tornar complexa. Não foram usados *frameworks* para injeção de dependências, apenas foram executadas as especificações de injeção de acordo com o JAVA EE 6[4].

IV. Arquitetura e Procedimentos

Antes de iniciarmos a implementação do projeto, viu-se necessário a criação de uma arquitetura a fim organizar os componentes do sistema e suas interações.

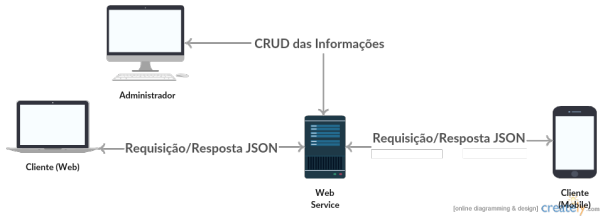

Figura 3. Arquitetura do sistema implementado

A figura 3 aponta três tipos de interações com o *Web Service*, sendo elas:

- A primeira interação é entre o controle administrativo e o *Web Service*.
- A segunda interação é entre a interface web e o *Web Service*.
- A terceira interação é entre o aplicativo móvel e o *Web Service*.

A. Cliente

Um dos principais objetivos de um *Web Service* é prover informações para ambientes, desde requisições de informações até envio de dados para serem armazenados. Para a implementação das ferramentas que o cliente pode usar para acessar o serviço proposto neste trabalho, foi usado o seguinte formato, tanto para a aplicação móvel quando a Web:

- Cadastro de cliente: tela de login com uma opção para que o usuário possa se cadastrar. Para este cadastro, o cliente deve informar um usuário e uma senha e, para salvar essas informações no *Web Service*, usou-se o método *POST* que, como resposta, retorna uma mensagem mostrando se o cadastro foi realizado com sucesso ou não. Então, se o cadastro for realizado com sucesso, o aplicativo retorna o usuário para a tela de login. Um ponto importante a se destacar é o fato de que a senha é convertida no formato *MD5*, para garantir uma melhor segurança no momento do armazenamento das informações dos clientes.
- Login do cliente: para o login, o usuário deve preencher os campos de usuário e senha. Assim, a aplicação envia um método *GET* para o *Web Service* que retorna como mensagem se as informações estão corretas e, caso esteja, o aplicativo redireciona o cliente para a tela inicial.
- Página inicial: nesta página, o cliente pode navegar entre as opções que o aplicativo oferece, tais como a tela para visualização dos filmes que estão em cartaz e os filmes que ainda serão lançados.
- Exibição dos filmes em cartaz: esta página mostra todos os filmes que estão sendo exibidos no cinema, bem como as informações como horário de exibição, sinopse, trailer, gênero, e se o filme é do tipo *3D* ou não. Além destas, a aplicação mostra uma lista de todas os comentários dos clientes e uma nota geral do filme.
- Exibição dos filmes que ainda serão lançados: está página foi implementada da mesma forma que a página de exibições de filmes em cartaz, a diferença é que esta não conta com as avaliações do cliente.
- Cadastro e visualização das avaliações dos filmes feitas pelos clientes: assim que o cliente seleciona um filme na seção de exibição de filmes em cartaz, o cliente pode atribuir uma nota e um comentário sobre filme. Essas duas informações são enviadas usando o método *POST* para que possam ser armazenadas e acessadas posteriormente.

1) Controle Administrativo

Para a funcionalidade completa de um *Web Service* na maioria das vezes é necessário uma base de dados para armazenar essas informações e que seja utilizado pelo servidor para propagar essas informações, sendo elas por *JSON* ou *XML*.

Na implementação percebeu-se que seria necessário uma base de dados, bem como uma página web na qual algum representante da empresa possa estar armazenando e disponibilizando informações de filmes em cartaz e também de filmes que em breve serão lançados. Deste modo, foi implementado um banco de dados utilizando a ferramenta *MySQL*[5]. O *Mysql* é um sistema de gerenciamento de banco de dados que utiliza a linguagem *SQL* como padrão. Atualmente é um dos sistemas de gerenciamento de banco de dados mais utilizado do mundo com em torno de 10 milhões de instalações.

A partir da escolha do sistema de gerenciamento de banco de dados, foi realizado o planejamento de como esses dados seriam guardados, assim efetivou a divisão das tabelas apresentadas a seguir.

- **Administrador:** tabela principal que consiste do nome do usuário e uma senha, a pessoa cadastrada nesta tabela tem privilégios para adicionar e remover todas as demais informações cadastradas nas tabelas abaixo.
- **Filmes em Cartaz:** utilizada para armazenar as informações dos filmes que estão em cartaz, os campos permitidos era nome do filme, sinopse, gênero, qual sessão que

[4]http://docs.oracle.com/javaee/6/tutorial/doc/giwhb.html

[5]https://www.mysql.com/

será transmitida, se o filme será exibido no formato *3D*, trailer e uma imagem de capa.

- **Filmes disponibilizados em breve:** tabela também utilizada para armazenamento de filmes, porém que estarão em cartaz em breve, sua diferença em comparação com a tabela filmes em cartaz, é que não possui sessões para serem transmitidas nem se serão transmitidas em três dimensões.
- **Usuários:** tabela para armazenamento de informações dos usuários que poderão estar comentando sobre os filmes e cartaz e também realizando uma votação para seu filme preferível.
- **Comentários:** tabela para armazenamento dos comentários que foram realizados na aplicação móvel e, também, no *website*.

A partir da criação da estrutura do banco de dados, iniciou-se a implementação das conexões com o banco de dados que está em um ambiente externo. Para estas conexões, foi utilizado a linguagem de programação *PHP*[6] que tem como uma usabilidade pequenas porções de comandos para realizar as conversas com banco de dados externo. No seguinte trecho de código, pode ser demonstrado como é realizado essa conexão.

```
<?php
include_once("base_dados");
/* Informacoes do Banco de Dados*/
$host = "endereco_ip_hospedado_base";
$username = "username";
$password = "senha";
$database = "nome_base_dados_mysql";
$table = "nome_tabela_mysql";
/* Autenticacao */
mysql_connect("$host","$username","$password")
        or die ("Impossivel conectar!");
mysql_select_db("$database")
        or die ("Database invalida!");
?>
```

Desta forma, a linguagem *PHP* faz as conexões com o servidor que está com o banco de dados instalado e, após isto, é feito a transição dos dados do *website* para o *MySQL*.

Porém, para que haja um melhor trabalho dos usuários da administração, é necessário a criação de um ambiente considerado como *"front-end"*, que nada mais é as interfaces gráficas para que os usuários a utilizem. Neste trabalho, foi utilizado a ferramenta *Bootstrap*[7], que é uma coleção de ferramentas para criação de *websites* e aplicações web utilizando o *HTML* e *CSS*.

Através da utilização do *Bootstrap*, foi desenvolvido primeiramente uma tela de login para o acesso à administração e um conjunto de abas para realizar as operações de remoção e acréscimos de filmes.

2) Aplicação Móvel

Como o trabalho é sobre *Web Service*, viu-se a necessidade de implementar vários tipos de aplicações clientes para mostrar que o sistema funciona independentemente. Uma dessas aplicações é um aplicativo *mobile*. Para desenvolver esta aplicação, usou-se a linguagem Java com a IDE *Android-Studio*[8](ferramenta para desenvolvimento de aplicações *mobile* para o sistema operacional *Android*). A aplicação tem como foco acessar as informações do *Web Service* no formato *JSON* e usá-las para popular o aplicativo.

3) Aplicação Web

Uma grande importância de um *Web Service* é prover que software e serviços se comuniquem por meio de intercâmbio de dados computacionais, isto é, mesmo que forem implementados de maneiras distintas os sistemas podem se comunicar de uma maneira rápida e compacta.

A aplicação web foi dividido em duas camadas sendo o *backend* e *front-end*. A camada de *backend* foi implementada na linguagem de programação *PHP*, esta camada foi responsável pela realização do preenchimento da camada mais acima que assegura a visualização para o usuário final. Para esta camada denominada *front-end*, foi utilizado um *template* gratuito disponibilizado na web para *websites* de cinemas implementado em linguagem *HTML* e *CSS*.

V. Resultados e Discussões

Ao fim de todas as implementações, foram concluídas quatro aplicações, sendo elas:

1) *Web Service Rest* com CDI;
2) Interface para o controle administrativo;
3) Aplicativo *Android*;
4) *Website*

O item 1 foi configurado e hospedado em um servidor da *Digital Ocean*[9]. Assim foi possível realizar todos os testes e requisições ao *Web Service* sem ter que testar localmente. Com acesso normal à internet, é possível acessar o*Web Service*, ou seja, a interface do controle administrativo, o aplicativo e o *website* podem se comunicar com métodos *GET* e *POST* normalmente.

A figura 4 mostra as três principais telas do Aplicativo *Android*, sendo que a imagem à esquerda mostra a tela de login e cadastro, a imagem do centro representa a lista de filmes que estão em exibição e a imagem à direita apresenta a tela onde o usuário avalia o filme.

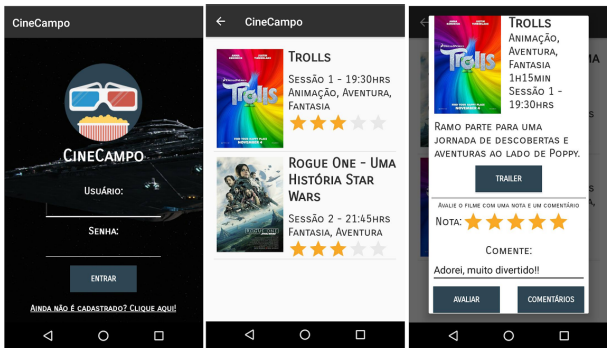

Figura 4. Telas do Aplicativo *Android*

Na figura 5 é mostrado qual é a interface de controle do administrador, sendo que, a imagem superior representa a tela

[6]https://secure.php.net/manual/pt_BR/index.php
[7]http://getbootstrap.com/
[8]https://developer.android.com/studio/index.html?hl=pt-br
[9]https://www.digitalocean.com/

de login do administrador e a imagem inferior representa a tela pós login em que o administrador pode cadastrar os filmes.

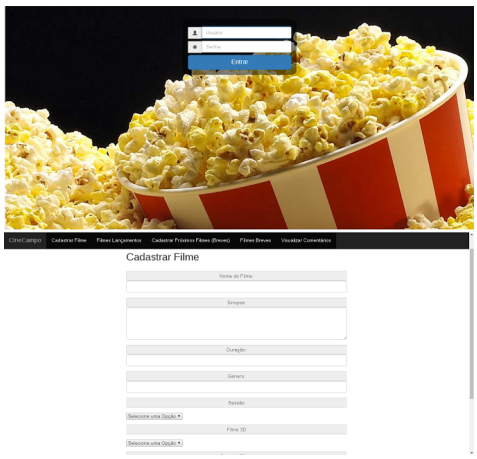

Figura 5. Interface do Controle Administrativo

Por fim, na figura 6, é exibido a interface do *website*, em que os usuários do cinema podem acompanhar os filmes que estão em cartaz ou os filmes que em breve estarão nos cinemas. A imagem superior representa a página inicial onde está cadastrado um dos filmes em cartaz e a imagem inferior mostra a aba em que o usuário avalia com uma nota e comentário sobre o filme em cartaz.

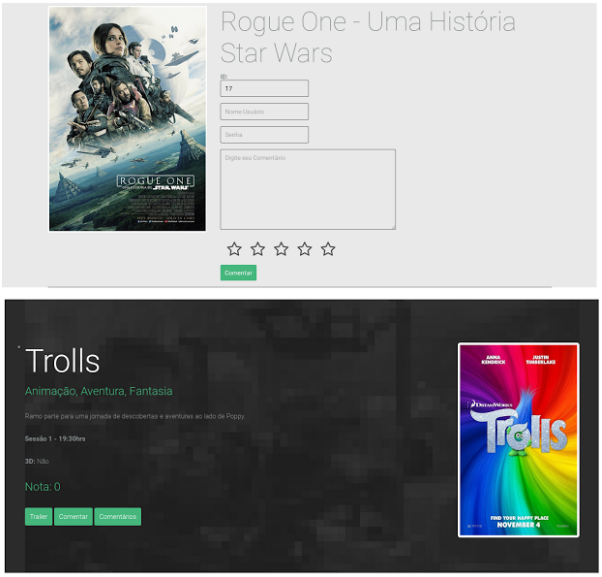

Figura 6. Interface Web

VI. Conclusão

Durante a implementação de todos os elementos que compõem o trabalho, alguns *frameworks*, servidores como o *TomCat* e *IDE's* como o *Eclipse* foram difíceis de serem configurados mas, assim que os ambientes de trabalho foram finalizados, foi possível desenvolver o trabalho e por em prática a maioria dos conteúdos da disciplina de Sistemas Distribuídos. Além disso, foi possível aprender algumas ferramentas e técnicas como a manipulação de arquivos *JSON*, injeção de dependência e contexto, *frameworks* para configuração de servidor, criação de *websites* e *apps*, configuração de hospedagem do *Web Service* no *Digital Ocean* e alguns conceitos de redes de computadores.

VII. Referências

[1] Sporny, M., Kellogg, G., and Lanthaler, M. (2014). Json-ld 1.0 - a json-based serialization for linked data. W3C Recommendation.

[2] Wilde, E. and Pautasso, C. (2011). Rest: from research to prac- tice.

[3] Seely, S. and Foreword By-Sharkey, K. (2001). SOAP: cross platform Web service development using XML. Prentice Hall PTR.