FACULDADE SÃO FRANCISCO DE ASSIS
CIÊNCIA DA COMPUTAÇÃO

Dener de Oliveira Kemele

Uma proposta de implementação de web service com servidor PHP e cliente Java para dispositivos móveis

Porto Alegre
2019

Dener de Oliveira Kemele

Uma proposta de implementação de web Service com servidor PHP e cliente Java para dispositivos móveis

Artigo apresentado à Faculdade são Francisco de Assis, como parte integrante dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Maurício de Oliveira Saraiva

Porto Alegre

2019

RESUMO

Uma das dificuldades é integrar sistemas em diferentes ambientes, e em diferentes linguagens de programação. Neste caso para resolver esta questão foram elaborados os Web Services, que fazem a comunicação de dados de forma independente. Este trabalho mostra a estrutura em ambiente web utilizando elementos estruturais para enviar e receber dados, utilizando uma arquitetura REST entre um servidor web desenvolvido em PHP e um cliente Java para dispositivos móveis. Também foram abordados outros elementos de comunicação, como protocolos e estilos, mantendo a estrutura de controle dentro do servidor PHP e a persistência em um bando de dados MySQL. Os resultados mostram a perfeita integração entre os sistemas de diferentes plataformas para a pesquisa de filmes pela Internet.

Palavras-chave: REST. Web services. Android. Java. PHP.

ABSTRACT

One of the difficulties is to integrate systems in different environments, and in different programming languages. In this case to solve this question were elaborated the Web Services, which do the communication of data independently. This work shows the structure in web environment using structural elements to send and receive data, using a REST architecture between a web server developed in PHP and a Java client for mobile devices. Other elements of communication, such as protocols and styles, were also addressed, maintaining the control structure within the PHP server and persistence in a MySQL database. The results show the perfect integration between systems of different platforms for the search of films by the Internet.

Keywords: REST. Web services. Android. Java. PHP.

1 INTRODUÇÃO

Nos tempos atuais existe a necessidade de utilizar a integração de sistemas para unificar comunicações em ambientes diferentes, visando facilitar a troca de dados entre diferentes plataformas.

Porém muitas organizações utilizam linguagens diferentes, gerando uma dificuldade quando se pretende obter informações de outros ambientes.

Para resolver esse problema foram criados web services, que são serviços que ficam na rede fazendo a troca de dado em vários ambientes, se tornando um integrador de dispositivos que estejam conectado à internet.

No decorrer do tempo foram surgindo diferentes meios para implementar esses ambientes de integração como o Representational State Transfer (REST), sendo um estilo arquitetônico, definindo padrões para o uso do protocolo HTTP. Também existe o Simple Object Access Protocol (SOAP), este é um protocolo que tem como objetivo a comunicação entre objetos e serviços.

Este artigo apresenta um ambiente de comunicação web, como o protocolo HTTP, sendo este implementado em um ambiente REST utilizando os verbos Get, Post, Put e Delete. Também apresenta várias estruturas de integração e conceitos como a respeito de web services, como SOA, SOAP, UDDI e WSDL.

1.1 Objetivos

O objetivo geral deste trabalho compreende a implementação de um Web Service, bem como o desenvolvimento de uma aplicação cliente para consumir os métodos desse serviço.

Para atender o objetivo geral, os seguintes objetivos específicos foram elencados:

- Reconhecer os conceitos e as características de cada tipo de Web Service;
- Configurar um banco de dados MySQL para persistir dados da aplicação;
- Implementar um Web Service em PHP e desenvolver os métodos de acesso;
- Desenvolver um aplicativo cliente em Java, que rode em dispositivos móveis Android, para consumir aquele método desse Web Service.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Hyper Text Transfer Protocol (HTTP)

Com grande crescimento em escala mundial a Web se tornou cada vez mais utilizada, tanto em pequenos grupos publicando seus tópicos e grandes empresas aumentando muitas mais sua capacidade de divulgação.

A comunidade de desenvolvedores da Internet ficou preocupada que o rápido crescimento do uso da Web, juntamente com algumas características ruins de rede do HTTP inicial, superasse rapidamente a capacidade da infraestrutura da Internet e levasse a um colapso geral (FIELDING, 2000).

Projetado na década de 1990, é um protocolo que evoluiu gradativamente. É um protocolo de comunicação da camada de aplicação que é enviado sobre TCP.

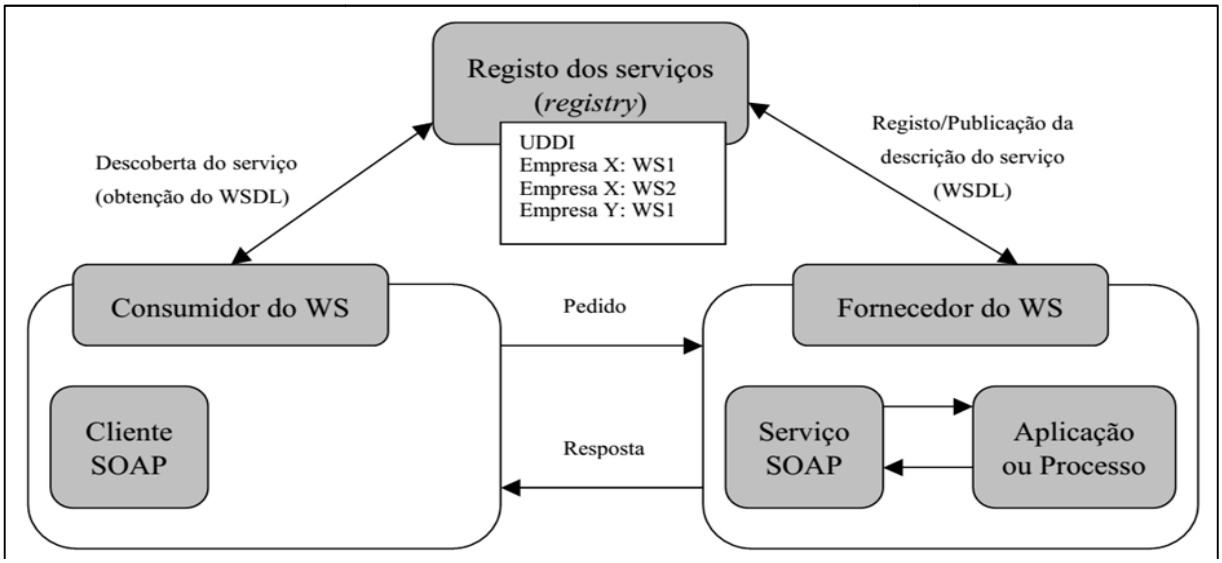
HTTP não possui estados, porém, tem seções, na mesma conexão não existe requisições simultâneas sendo feitas. O principal fator do HTTP é não manter os estados, porém, os cookies HTTP permitem que as sessões iniciadas tenham estados, assim esses cookies são adicionados ao fluxo HTTP, cada requisição possui uma sessão compartilhando o mesmo contexto ou o mesmo estado.

2.2 Service-Oriented Architecture (SOA)

SOA é um estilo de arquitetura de software em que as aplicações devem ser disponibilizadas na forma de serviços. Tais serviços são disponibilizados na web, pois a implementação SOA utilizam Web Services.

Uma das maneiras de explicar SOA seria *separação de interesses*, essa teoria propõe essencialmente que os problemas maiores sejam decompostos em uma série de problemas menores individualmente identificáveis. A lógica necessária para resolver o problema maior pode, então, ser também dividida em unidades individuais de lógica que tratam de preocupações específicas (ERL, 2005).

Figura 1: Estrutura SOA



Fonte (PINHO, 2008, p. 14)

■ Requisitante de serviços (Cliente): O fornecedor e o consumidor do WS têm de se *conhecer* mutuamente ou, pelo menos, um deles tem de *conhecer* o outro.

■ Provedor de serviços (Servidor): Os fornecedores do WS têm de chegar a acordo relativamente à descrição e semântica que serão aplicadas às interações.

■ Broker de serviços (Registro dos serviços - UDDI): A descrição e semântica do serviço têm de ser compreendidas por ambos os agentes (o cliente *contrata* ao fornecedor do WS esse serviço, ficando apto a poder usá-lo). A informação contida na descrição e semântica do serviço deve ser introduzida e implementada em cada um dos agentes (PINHO, 2008).

2.2.1 Princípios comuns do SOA

Existem oito princípios comuns da arquitetura orientada a serviços:

■ **Contrato de serviço padronizado.** Todos os dados devem possuir padrão para poderem ser disponibilizados na solicitação, assim facilitam a procura dos dados, pois é fundamental para a interação ente o serviço e o consumidor. As unidades de lógica de automação classificadas como serviços devem fornecer um contrato no qual os termos de compromisso são definidos. Como um todo, os contratos de serviços podem ser compostos por informações legais e técnicas (ERL,

2005). O Contrato de serviço permanece imutável, de modo a preservar as relações estabelecidas de serviço-cliente.

■ **Baixo acoplamento.** Minimiza as dependências entra componentes assim facilitando a migração de plataformas, bem como a engenharia de serviços. Tais serviços não são fortemente acoplados nem são desacoplados, são classificados como soltos. As operações de serviços da Web fazem parte de um *contrato de serviço* que abstrai os detalhes da funcionalidade que elas operam.

Neste caso, acessa-se um ponto de uma interface que direciona para o componente que vai oferecer o serviço, dentre todos que estão acoplados.

■ **Abstração de serviços**. A abstração dos serviços permite que qualquer alteração terá um impacto mínimo para o cliente. Assim o cliente terá acesso somente aquilo solicitado referente ao serviço e não a implementação.

■ **Reutilização do serviço.** Reutilizar o serviço em outros escopos, utilizando uma lógica genérica assim como resultado, a reutilização de serviços é um princípio quase sempre realizado por meio de projetos padronizados. As principais características do serviço que apoiam e ajudam a promover a reutilização são a autonomia, sem estado e a descoberta (ERL, 2005).

■ **Autonomia do serviço.** O serviço tem que ser independente, sem resolvendo sozinho sem precisar de dependência de outros recursos. Os serviços puramente autônomos têm a propriedade absoluta de seus recursos, o que permite que eles sejam mais bem ajustados para eficiência, confiabilidade e disponibilidade. (ERL, 2005, p. 3).

■ **Serviços sem Estado.** O tempo de vida do serviço é associado a tarefa, sendo a manipulação e retenção de dados relacionado ao estado devem ocorrer no tempo de execução. Um gerenciamento de estados consome muitos recursos, assim manter uma condição sem estado beneficia um serviço aumentando a disponibilidade e escalabilidade. Observando que, diferentemente da autonomia, os serviços não podem atingir um nível de total sem estado. Durante o processamento, um serviço possui um estado por um período. Este princípio simplesmente enfatiza que a duração desse período deve ser minimizada (ERL, 2005).

■ **Descoberta de serviços.** Os contratos de serviço descrevem aspectos técnicos de um serviço para facilitar o consumo de clientes em potencial. Especificamente, a capacidade de descoberta introduz convenções que promovem clareza e descrição em um nível de pequenas partes (ERL, 2005). Para reutilização

ser possível somente se a informação puder ser descoberta em primeiro lugar, mantendo as informações consistentes e mantendo eles em um repositório pesquisável, assim permitindo que outras pessoas procurem as informações de maneira eficiente.

- **Modularidade.** O Serviço deve ser composto por vários outros módulos que são unidos para executar coletivamente uma tarefa específica.

Ao garantir que os serviços sejam capazes de participar de várias composições, um inventário de serviços adaptativos pode ser acumulado (ERL, 2005).

2.3 Application programming interface (API) / Web service

São sistemas que podem ser encontrados na internet com um caminho, estando expostos na rede, gerando uma comunicação entre esses sistemas.

Com a utilização do SOA visando organizar recursos, que podem estar em diferentes organizações, gerando um meio padronizados. A integração utilizando SOA, tem como objetivo a integração em diversos sistemas. Assim um meio de implementar é utilizando a Integração orientada a Serviços (SOI) e tem como principal objetivo a integração de diversos sistemas modificando pouco ou nada suas implementações assim utilizando web services, pois possuem uma interface de serviços que possibilita a interação de consumidores e prestadores de serviço.

Assim o Web service tendo dois tipos de utilização, sendo o mais regular como provedor de serviços, onde um sistema posicionado em outro local realiza solicitação, sendo atribuída o nome de *request*. Após ler a solicitação o Web Service processa o pedido e devolve as informações seguindo um padrão de envio.

2.4 Simple Object Access Protocol (SOAP)

O SOAP é um protocolo leve destinado a trocar informações estruturadas em um ambiente distribuído e descentralizado. Ele usa tecnologias XML para definir uma estrutura de mensagens extensível, fornecendo uma construção de mensagem que pode ser trocada através de uma variedade de protocolos subjacentes (W3C, 2007).

Sendo utilizado pela W3C como padrão industrial, tendo fácil utilização e implementação. A partir do protocolo HTTP as mensagens são transportadas por pacotes idênticos, sendo um protocolo que utiliza aplicações remotas através de *Remote Call Procedure* (RCP) ou trocas simples de mensagens, todavia, independe de qualquer plataforma e linguagem de programação.

2.4.1 Formato de mensagem

O XML *information set*, é uma especificação do W3C para um documento XML abstrato, foi escolhido porque grandes corporações utilizavam e, também, por ser em código aberto. O XML pode trazer vários benefícios como a interoperabilidade, bem como a detecção de erro, e um problema por ser mais extensa pode diminuir a velocidade de processamento.

A comunicação típica para o SOAP é HTTP, podendo usar os métodos GET ou POST. Com GET, a solicitação não é uma mensagem SOAP, mas a resposta é uma mensagem SOAP, já com POST a solicitação e a resposta são mensagens SOAP.

SOAP usa os mesmos erros e status dos usados em HTTP, assim as respostas podem ser interpretadas diretamente por um Módulo SOAP.

Uma mensagem SOAP é um XML composto por um envelope, um cabeçalho, um corpo e uma falha:

- **Envelope**. Identificação do documento XML, sendo necessária. encapsula a mensagem SOAP;
- **Cabeçalho**. Contém informações de cabeçalho, sendo opcional;
- **Corpo**. Contém informações de chamada e resposta, sendo necessária;
- **Falha**. Contém informações sobre erros, fazendo parte do corpo. Sendo opcional.

Vantagens do uso do SOAP:

- Pode ser utilizado em qualquer protocolo de transporte;
- Por utilizar HTTP na porta 80 facilita na passagem por firewalls ou proxies;
- Possui todas as facilidades do XML.

Desvantagens do uso do SOAP:

- Perde de desempenho na serialização em XML;
- Não possui sistema de endereçamento de WS.

2.5 Web Services Description Language (WSDL)

WSDL é um documento em formato XML que descreve as regras de uma web service, descrevendo informações importantes, como os métodos, locais e os elementos principais.

As definições de serviço especificam um contrato básico entre o provedor de serviços e quaisquer possíveis clientes, detalhando os tipos de funções que são fornecidas pelo serviço e as mensagens trocadas como parte de cada função. Os provedores de serviços e consumidores são livres para implementar as suas extremidades da troca como quiserem, desde que as mensagens que eles enviam correspondam à definição de serviço (SOSNOSKI, 2011).

Uma estrutura de um arquivo no formato WSDL possui quatro elementos principais segundo (SOSNOSKI, 2011): *types, messages, portType e binding*:

- **Types.** Descreve os tipos de dados que serão usados entre cliente e servidor;
- **Message.** Define os elementos de entrada e saída sendo identificando se é *request*/*response*;
- **PortType.** Define as mensagens e as operações envolvidas;
- **Binding.** Tem o objetivo de nomear a operação disponibilizada pelo Web service. Quatro tipos de operações que podem ser usadas: notification, solicit-response, request-response, one-way.

2.6 Universal Description, Discovery and Integration (UDDI)

UDDI é uma aplicação de diretório onde as pessoas podem publicar ou buscar serviços de Web services. Sendo projetado para ser usado por mensagem SOAP e fornecer acesso a documentos WSDL, também se baseando em vários outros padrões estabelecidos do setor, incluindo HTTP, XML, Esquema XML (XSD).

No contexto de Arquitetura Orientada a Serviços o UDDI tem a função de Brokers de serviço, que permite o consumidor localizar um provedor de serviços. Estrutura de um documento padrão UDDI é descrita por:

- **MNodelo de dados businessEntity.** Contém informações sobre o negócio que tem um serviço publicado;
- **Modelo de dados businessService.** É uma descrição de um serviço Web, mostrando quem realiza o serviço.
- **Modelo de dados bindingTemplate.** Contém informações técnicas para determinar o ponto de início e as especificações de construção para chamar um serviço Web.
- **Modelo de dados tModel.** Fornece um sistema de referência para ajudar na procura de serviços Web e atua como uma explicação técnica para um serviço Web.

2.7 Extended Markup Language (XML)

Criada em 1997 pela W3C, para operar e levar dados de modo mais ágil, e uma arquitetura que possui elementos e marcas predefinidas. Não diz como os autores vão manusear os metadados, sendo que existe total liberdade para lidar com qualquer método disponível, desde simples atributos, até a implementação de padrões mais relevantes (BARCELLOS, 2007).

Os Elementos ‘<’ ‘>’ para início e final, respectivamente, são chamados de marcadores ou tags, sendo componentes essenciais de um XML. Os Elementos podem possuir outros elementos dentro do mesmo, bem como textos, sempre iniciando e fechando para não haver erro de sintaxe.

2.8 Representational State Transfer (REST)

A lógica de projeto por trás da arquitetura da Web pode ser descrita por um estilo arquitetural que consiste no conjunto de restrições aplicadas a elementos dentro da arquitetura. Examinando o impacto de cada restrição à medida que ela é adicionada ao estilo em evolução, podem-se identificar as propriedades induzidas pelas restrições da Web (FIELDING, 2000).

REST é um estilo híbrido, formado de outros estilos arquitetônicos, são estilos criados para as necessidades que melhoram as propriedades de comunicação ou interação de aplicações baseadas em rede, sendo alguns estilos:

- Estilo de fluxo de dados;
- Estilo de Replicação;
- Estilo Hierárquico;
- Estilo Código móvel;
- Estilo Peer-to-Peer.

2.8.1 Restrições do REST:

REST é formado por sete restrições que definem a arquitetura:

- **Estilo Nulo ou vazio.** É o estilo raiz onde não há nenhuma restrição, sedo um conjunto vazio que descreve que não tem limite entre os componentes.
- **Cliente-Servidor.** Separa os interesses do cliente e servidor, assim tratando cada uma em seu lado, aprimorando e simplificando os componentes do servidor.
- **Sem Estado.** Todas as informações de solicitação devem estar no cliente e não pode aproveitar nenhum contexto do servidor. O estado de sessão é mantido inteiramente no cliente. Assim como os dados não são armazenados no servidor então a aplicação quando for fazer um novo processo igual a algum anterior terá que fazer de novo uma nova solicitação com todas informações, podendo gerar uma sobrecarga.
- **Cache.** Restrição na utilização de alguns dados de cache para melhorar as interações em solicitações equivalentes. Podendo haver dados obsoletos dentro do cache.
- **Interface Uniforme.** Uniformidade dos componentes possuindo uma arquitetura simplificada e a visibilidade das alterações é aprimorada. As informações são transferidas de forma padrão ao invés das necessidades da aplicação.
- **Sistema em camadas.** Camadas hierárquicas restringem o comportamento dos componentes, vendo somente suas camadas que estão interagindo, assim garantindo um balanceamento de carga.
- **Código sob demanda.** O cliente baixará previamente scripts para agilizar o processo.

2.9 JavaScript Object Natation (JSON)

Para resolver vários problemas referente a troca de informações entre sistemas, foram desenvolvidas algumas estruturas de arquivos, sendo JSON uma das mais leves e usadas. JSON pode representar quatro tipos primários (*strings*, números, booleanos e nulos) e dois tipos estruturados sendo objetos e vetores (FONSECA, 2007). Possuindo fácil serialização e transporte de dados, assim com seu fácil entendimento, cria uma vantagem sobre outros objetos de notação. Assim, JSON foi escrito com o objetivo de ser simples, fácil e textual e um subconjunto do JavaScript.

3 MÉTODO

Para desenvolver este trabalho uma pesquisa bibliográfica foi realizada nos mecanismos de busca em bibliotecas e repositórios virtuais sobre os seguintes temas: Web service, banco de dados e desenvolvimento de aplicativos para dispositivos móveis. Nessa pesquisa as seguintes palavras-chave foram utilizadas: REST, SOAP, XML, JSON, Web Service, SOA, HTTP, WSDL, MVC, Android, Java, PHP, MySQL, na qual diversos trabalhos foram encontrados.

A pesquisa realizada formou uma base de conhecimento que permitiu realizar a fundamentação teórica deste trabalho, bem como definir as tecnologias e os recursos utilizados para a implementação de um Web Service que acessa um banco de dados em conjunto com o desenvolvimento de um aplicativo cliente para dispositivos móveis.

Além da pesquisa citada, este trabalho contou com o desenvolvimento de uma aplicação prática, contendo a modelagem de um banco de dados MySQL, a implementação de um Web Service REST em PHP com framework MVC e o desenvolvimento de um aplicativo cliente para dispositivos móveis, implementado em Java.

4 IMPLEMENTAÇÃO

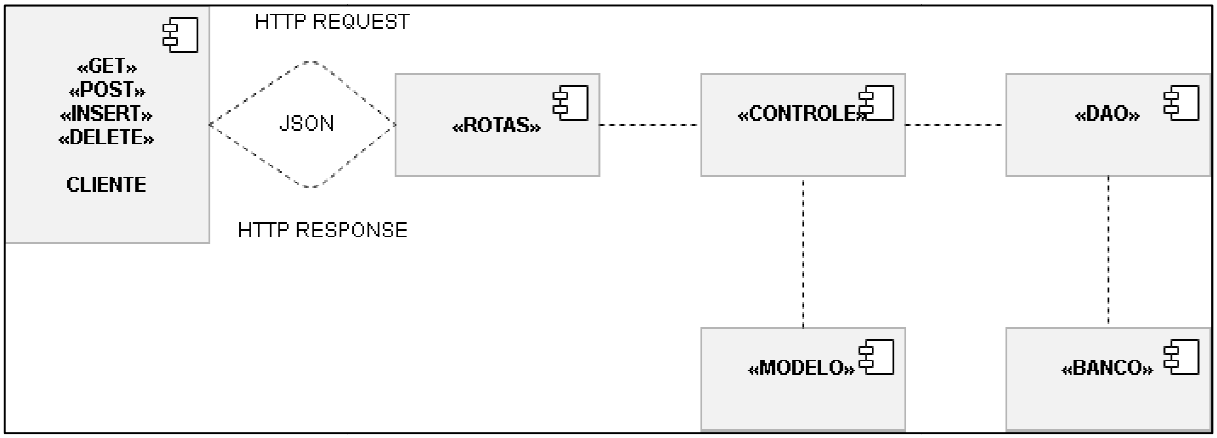
4.1 Arquitetura

A arquitetura é baseada no modelo Cliente/Servidor em duas camadas (*Model*/*Controller*), já que um Web Service não possui a camada de apresentação (View) para exibir dados.

Os clientes podem consumir os métodos do Web Service por meio de requisições (*requests*) que são tratadas pelas rotas de direcionamento. A camada *Controller* se comunica com a camada *Model* para manipular os dados de entrada e assim para acessar a camada *DAO* que mapeia os objetos do banco a partir de consultas SQL.

No retorno (*response*), o Web Service envia os dados ao cliente em formato JSON, contendo as informações que estão armazenadas no banco de dados, conforme ilustrado pela Figura 2.

Figura 2: Arquitetura



Fonte: Elaborador pelo autor

Definição de rotas, controle e modelo:

- **Rotas.** Define o caminho que o cliente vai seguir, pois utiliza verbos HTTP sendo GET, POST, PUT e DELETE. Para cada rota um método de uma classe vai ser executado.

- **Controle.** É um conjunto de classes sendo este responsável por receber e enviar as mensagens e fazer o tratamento em JSON.

■ **Modelo.** Este faz o tratamento de valores se comunicando com o Controle e camada DAO.

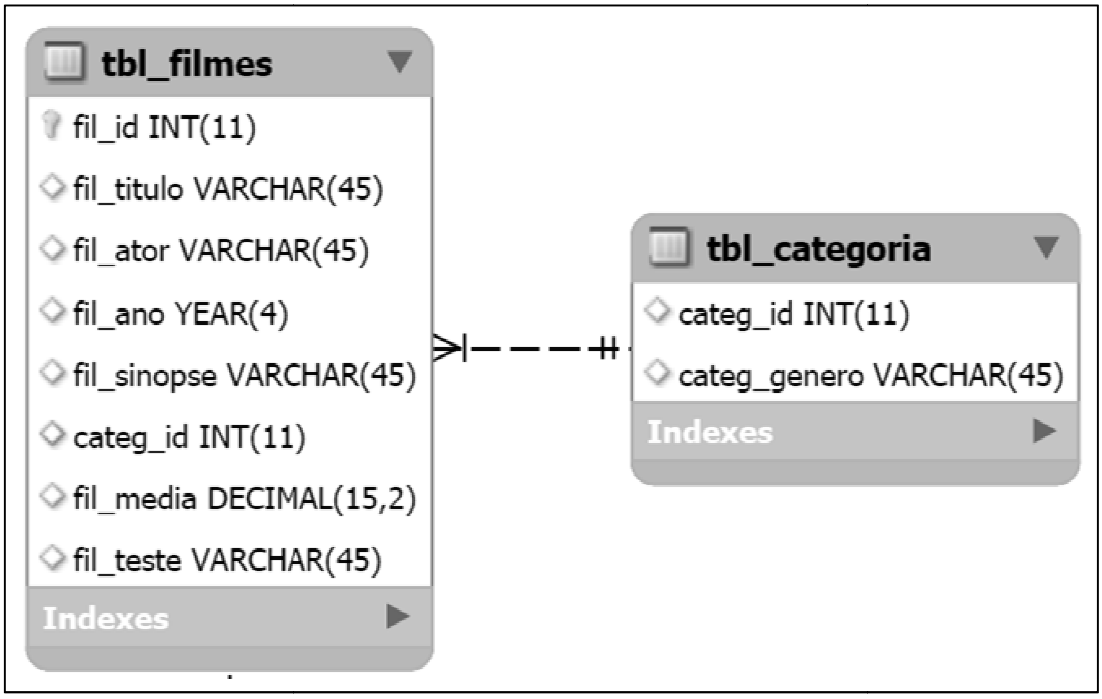
■ **DAO.** Segue o *design pattern* que faz o acesso e a manipulação das informações no banco de dados.

4.2 Banco de dados

Um banco de dados relacional foi modelado em MySQL versão 5.0 para armazenar algumas informações sobre filmes, como título, autor, ano, sinopse e categoria. Essas informações foram definidas em duas tabelas relacionadas: tbl_filmes e tbl_categoria, conforme ilustra a Figura 3.

O relacionamento entre as tabelas está definido pela chave estrangeira *categ_id* da tabela tbl_filmes, que é chave primária na tabela tbl_categoria. Índices foram criados na tabela tbl_filmes para obter desempenho nas pesquisas por autor e título, que são os campos mais pesquisados.

Figura 3: Diagrama Entidade Relacionamento (ER)

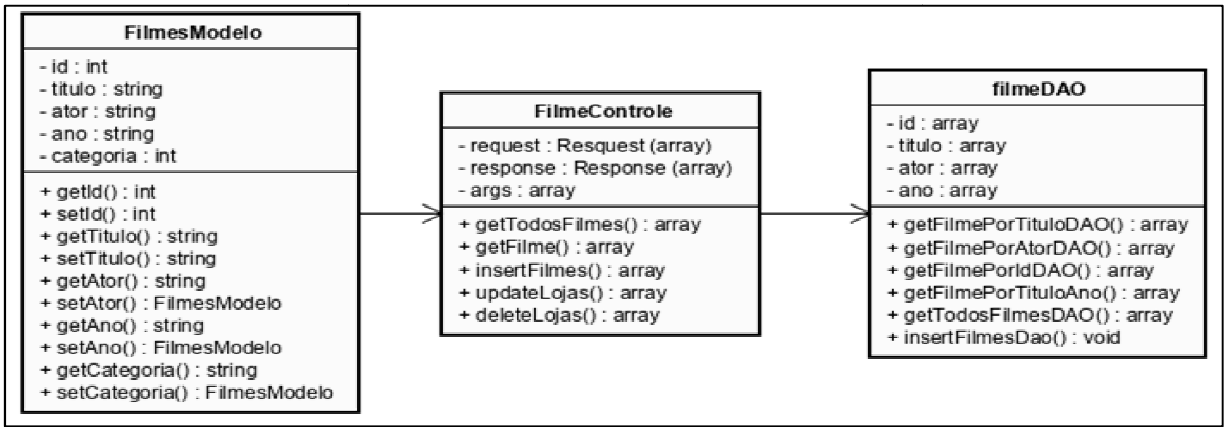


Fonte: Elaborado pelo autor

4.3 Diagrama de classe

No diagrama apresenta todos os métodos a atributos das funções implementadas, assim separadas por classes. Cada uma faz uma interação referente a camada que ela se encontra podemos ser na classe modelo, controle ou DAO. “Os diagramas de classes descrevem a estrutura estática de um sistema, em particular as entidades existentes, as suas estruturas internas, e relações entre si.” (MANUEL; VIDEIRA, 2001, p. 125).

Figura 4: Diagrama de Classes



Fonte: Elaborado pelo autor

4.4 Servidor Web Service

A linguagem de programação utilizada para implementar o Web Service foi PHP versão 7.3, rodando o framework Slim na versão 3 para mapear rotas e acessar o banco de dados MySQL por meio de Data Access Objects (DAO).

A implementação do sistema foi feita sobre o *framework* Slim, sendo um *framework* que auxilia na escrita de aplicações Web. Ele fornece um roteador que mapeia o retorno das camadas de rotas para métodos HTTP.

Inicialmente instanciam-se as rotas que serão seguidas pela aplicação, sendo uma rota para cada Método Get, Post, Put e Delete. Cada rota fará a chamada de uma função da classe FilmeControle.

```
1 $var->get('/filmes[/{id}]', FilmeControle::class . ':getFilme');
2 $var->post('/filmes', FilmeControle::class . ':insertFilmes');
3 $var->put('/filme', FilmeControle::class . ':updateFilmes');
4 $var->delete('/filme', FilmeControle::class . ':deleteFilmes');
```

Na classe FilmeControle serão instanciadas as funções que manipularão o banco de dados, sendo: pesquisa, alteração, inserção ou exclusão. Cada função possui três parâmetros:

- Request. Traz as informações vindas do cliente, como os dados dos filmes;
- Response. Leva a reposta que o servidor vai enviar para o solicitante;
- Dados. Array de argumentos com informações de parâmetros da URL.

O acesso aos dados é realizado pela classe FilmeDao, que se conecta diretamente no banco de dados estendendo a classe conexão presente na estrutura do Web service.

As funções desta classe utilizarão instruções SQL para manipular o banco de dados por meio do método PHP Data Objects (PDO).

```
$sql = ('SELECT
  fil_ator as ator,
  fil_titulo as titulo,
  fil_ano as ano,
  fil_sinopse as sinopse,
  fil_categoria as categoria
  FROM tbl_filmes WHERE fil_titulo = :titulo ;');
  $stmt = $this->pdo->prepare($sql);
  $stmt->execute([":titulo" => $titulo]);
if($stmt->rowCount() > 0) {
    $filme = $stmt->fetchAll(\PDO::FETCH_ASSOC);
} else{
    $filme = [(["menssagem" => "Filme Nao existe"])];
}
return $filme;
```

Retornando para classe Filme a resposta será convertida em JSON utilizando a função withJson. E sendo enviado para o cliente solicitante via o método HTTP.

```
$response = $response->withJson($filme);
return $response;
```

A classe FilmesModelo é utilizada para definir os tipos de valores que serão inserindo no banco de dados, e configurando cada parâmetro para um tipo de dado. Depois chamado a classe DAO e pela função insertFilmesDao inserem-se os parâmetros nas colunas e tabelas especificas no banco de dados. Por fim, a reposta que será convertida em JSON.

```
$filmes = new FilmesModelo();
$filmes->setTitulo($data['titulo'])
       ->setAtor($data['ator'])
       ->setAno($data['ano'])
       ->setSinopse($data['sinopse']);
$filmesDAO = new filmeDAO();
$filmesDAO->insertFilmesDao($filmes);
$response = $response->withJson(["mensagem" => "Dados inseridos" ]);
return $response;
}
```

4.5 Aplicação cliente

A tecnologia escolhida foi Java voltada para aplicações moveis, sendo esta desenvolvida para o sistema operacional Android.

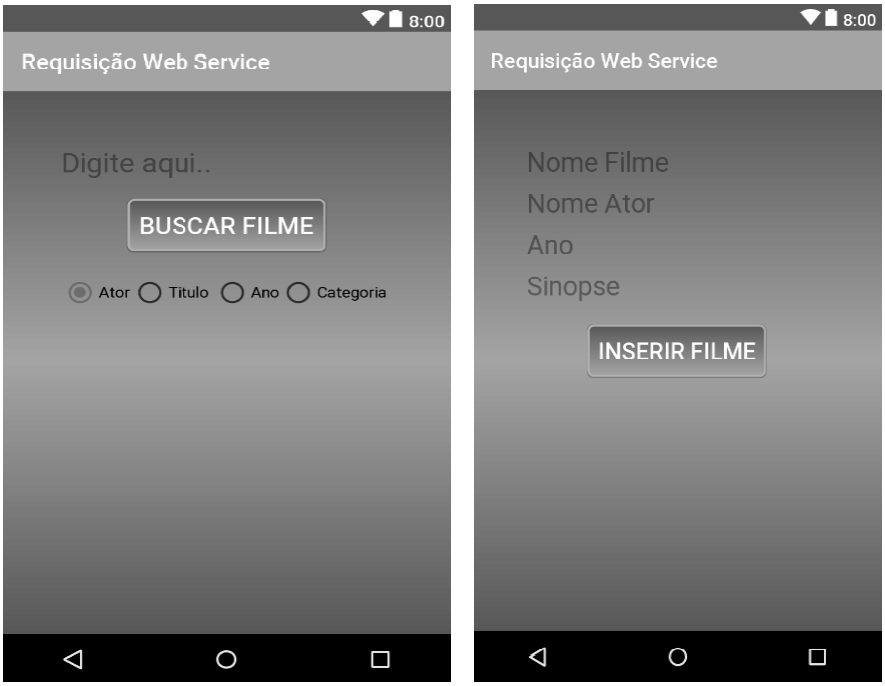
O cliente possui três telas principais, MainActivity, TelaCadastro e TelaPesquisa. Duas classes de comunicação HTTP com Web Service, HttpServiceGet e HttpServicePost e três telas de layout, sendo activity_main, activity_tala_cadastro, activity_tela_pesquisa.

A aplicação é formada por três telas de interação: a primeira é a tela principal que é formada por dois botões que acionaram as telas de pesquisa de filmes ou cadastros de filmes.

A segunda é a tela de pesquisa tem a opção de colocar o nome do item a ser pesquisado e, contudo, pode este item ser o nome do filme, o ator do filme, o ano de lançamento e a categoria, apenas selecionando a procura em questão.

A terceira tela é formada pelos campos de textos para inserção de valores sendo o nome do filme, do ator, do ano e a sinopse, conforme ilustrado na Figura 5.

Figura 5 Telas de pesquisa e de resultado



Fonte: Elaborado pelo autor

A tela MainActivity é a primeira classe a ser instanciada e está sendo implementada pela função OnClickListener servindo para o evento do click e chamando a interface criando os métodos onCreat e.

```
1 public class MainActivity extends AppCompatActivity
2       implements View.OnClickListener {
3 }
```

Sendo a instancia onCreat para criar os métodos para tela de layout e a ação dos eventos botões e onClick redireciona a ação do evento do click dos botões para tela de pesquisa ou tela de cadastro.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
4      super.onCreate(savedInstanceState);
5      setContentView(R.layout.activity_main);
7      Button btn_pesquisar = (Button) findViewById(R.id.btn_pesquisa);
8      btn_pesquisar.setOnClickListener(this);
10     Button btn_inserir = (Button) findViewById(R.id.btn_inserir);
11     btn_inserir.setOnClickListener(this);
13 }
```

Na tela de pesquisa é feita a instancia dos eventos a serem pesquisados, assim fazendo a chamada da classe HttpServiceGet onde enviara os dados e fara a comunicação HTTP com o servidor que gerenciara os pedidos.

```
1     new HttpServiceGet(nome.getText().toString(), var1, var2, var3, var4);
```

A classe HttpServiceGet é chamada e assim primeiramente define qual rota será feita, sedo feita uma requisição via GET é enviada uma String anexada na URL, todavia dependendo da pesquisa solicitada trazendo os campos fornecidos pela tela de pesquisa.

```
1  if (this.ator == 1) {
2      new HTTPAsyncTask().execute("
3       http://192.168.15.6/webservice/filmes?ator=" + this.nome);
4  } else if (this.titulo == 1) {
5      new HTTPAsyncTask().execute("
6      http://192.168.15.6/webservice/filmes?titulo=" + this.nome);
7  } else if (this.ano == 1) {
8      new HTTPAsyncTask().execute("
9      http://192.168.15.6/webservice/filmes?ano=" + this.nome);
10 } else if (this.categoria == 1) {
11     new HTTPAsyncTask().execute("
       http://192.168.15.6/webservice/filmes?categoria=" +
12 this.nome);
13 } else {
14     new HTTPAsyncTask().execute("
15     http://192.168.15.6/webservice/filmes");
16 }
```

Depois que definida qual vai ser a rota e feita a chama da função HTTPAsyncTask que vai estender a classe AsyncTask que fara a execução da do pedido em uma Thread separado para não travar a tela do usuário assim utilizando o método desta classe sendo o doInBackgroundque faz todo trabalho pesado da Thread.

```
1  private clas HTTPAsyncTask extends AsyncTask<String, Void, String> {
2     @Override
3      protected String doInBackground(String... urls) {
4  }
```

Depois cria-se a conexão com o método openConnection() da classe HttpURLConnection. Assim definindo o método de envio que nesse caso vai ser o GET pois só está sendo solicitado uma leitura de dados, e também o tipo de dados que vai ser recebido.

```
1 HttpURLConnection conn = (HttpURLConnection) url.openConnection();
2 conn.setRequestMethod("GET");
3 conn.setRequestProperty("Content-Type", "application/json;
4   charset=utf-8");
```

Em seguida o servidor enviar uma resposta para solicitação e está reposta será no formato JSON, então os dados da Stream são armazenados em um buffer para serem tratados.

```
1 InputStream = conn.getInputStream();
2 BufferedReader = new BufferedReader(
3    new InputStreamReader(inputStream));
```

Assim este array recebido será lido pela função JSONArray, depois utilizam-se a função JSONObject para pegar valor atribuído por uma chave do objeto JSON.

```
1  JSONArray JA = new JSONArray(data);
2  Serializable id;
4  for (int i = 0; i < JA.length(); i++) {
6    JSONObject JO = (JSONObject) JA.get(i);
7    if (JO.has("ator") || JO.has("titulo") || JO.has("sinopse")
8     || JO.has("categoria")) {
9     id = (i < 10) ? ("0" + i) : i;
11    singleParsed = id +" | Ator: " + JO.get("ator") + "\n" +
12    " | Título: " + JO.get("titulo") + "\n" +
13    " | Sinopse: " + JO.get("sinopse") + "\n" +
14    " | Categoria: " + JO.get("categoria") + "\n"+
16 } else
17       singleParsed = JO.get("menssagem") + "\n";
19   dataParsed = dataParsed + singleParsed + "\n";
20 }
21 return dataParsed;
22 }
```

No fim desse processo é executado a função onPostExecute da classe AsyncTask que vai enviar para tela de pesquisa a resposta da solicitação sendo convertida em texto.

```
protected void onPostExecute(String dataParsed) {
    super.onPostExecute(dataParsed);
    TelaPesquisa.data.setText(dataParsed);
```

Na tela de cadastro é feita a instancia dos eventos a serem cadastrados, assim fazendo a chamada da classe HttpServicePost onde enviará os dados e fara a comunicação HTTP com o servidor que gerenciara os pedidos de inserção.

```
new HttpServicePost(nome.getText().toString(),
    ator.getText().toString(),
    ano.getText().toString(),
    sinopse.getText().toString());
```

Na classe HttpServicePost segue o mesmo processo da tela de pesquisa porem como o método a ser utilizado é POST onde a requisição é encapsulada junto ao corpo da requisição HTTP e não pode ser vista. Então os dados são encapsulados e envia para a raiz da URL onde terá uma rota no servidor para tratá-lo.

```
HTTPAsyncTask().execute("http://192.168.15.6/webservice/filmes");
```

Seguindo o mesmo processo da Classe HttpServiceGet, ira estender a classe AsyncTask para trabalhar com um Thread separada invocando o método doInBackground. Assim ele chara a função para criar a conexão e desta vez será utilizado o método POST.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json");
```

Em seguida será construído um JSONObject que invocara a função buidJsonObject onde tratara os dados recebidos do formulário transformando em um objeto acumulado de JSON. Depois o JSON será incluído no corpo da solicitação POST. Por último e feito se conecta no URL definida via POST com a função connect.

```
 1 private JSONObject buidJsonObject() throws JSONException {
 3     JSONObject = new JSONObject();
 4     jsonObject.accumulate("titulo", nome);
 5     jsonObject.accumulate("ator", ator);
 6     jsonObject.accumulate("ano", ano);
 7     jsonObject.accumulate("sinopse", sinopse);
10     return jsonObject;
11 }
 1 JSONObject jsonObject = buidJsonObject();
 2 setPostRequestContent(conn, jsonObject);
 3 conn.connect();
```

Por fim faz o chamado da função getResponseCode e verifica o código de retorno, sendo 200 para certeza dos dados inseridos ou qualquer outro código para dados que não forem inseridos.

```
1 int statusCode = conn.getResponseCode();
2 messageCode = (statusCode == 200)? "Dados inseridos com sucesso" :
3  "Dados não inseridos";
```

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um sistema de web Service para integrar sistemas diferentes, sendo este num ambiente de comunicação REST, utilizando as linguagens de programação e PHP e Java. Primeiramente foi constatado a necessidade de as organizações precisarem comunicar sistemas que estão em estações de trabalho antigas a novas estações com sistema mais modernos. Com tal constatação mostrou que existem alguns tipos de sistema integrados web como o protocolo SOAP e os estilo de arquitetura REST. Todavia o mais utilizado tanto para mobile e outras aplicações, sendo o estilo REST por ser mais rápido e leve pois utiliza a estrutura HTTP para enviar dados leves como o JSON abordados no artigo.

Foi construído um banco de dados MySql estrutural para suprir a necessidade da aplicação assim podem armazenar dados e também pode-los pesquisar, funcionando para todas as solicitações requeridas.

O servidor que foi desenvolvido utilizou o Slim Framwork, que criou rotas de acessos para cada requisição do cliente, assim gerando todo o fluxo HTTP, e mostrou-se totalmente eficaz no controle e conversão dos dados.

O cliente por sua vez consumiu os dados vindos do servidor e assim tratando e apresentando para o cliente, com todas suas funções de pesquisa e inserção de dados, e interagindo muito bem no fluxo HTTP.

Para futuros trabalhos, se indica a pesquisa na parte de segurança trazendo métodos de autenticação tantos por toquem ou por usuário e senhas.

REFERÊNCIAS

BARCELLOS, R. A. **Uma introdução ao XML, sua utilização na Internet e alguns conceitos complementares**. Disponível em: <http://www.scielo.br/pdf/ci/v31n2/12903>. Acesso em: 10 jun. 2019.

ERL, T. **Informativo exclusivo do SOA Web Services Journal - Thomas Erl On SOA**. Disponível em: <http://soa.sys-con.com/node/136190/>. Acesso em: 29 out. 2018.

FIELDING, R. T. **Estilos arquitetônicos e o design de arquiteturas de software baseadas em rede**. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/web_arch_domain.htm>. Acesso em: 10 jun. 2019.

FONSECA, R. A. S. **Alternativas ao XML: YAML e JSON**. Disponível em: <http://repositorium.sdum.uminho.pt/bitstream/1822/6230/1/xmlyamljson07.pdf>. Acesso em: 10 jun. 2019.

MANUEL, A. S.; VIDEIRA, C. A. E. **UML, Metodologias e Ferramentas CASE**. Lisboa, p.125, 2001.

PINHO, S. C. **Arquitectura de Segurança num ambiente SOA (Service Oriented).** Disponível em: <https://repositorio-aberto.up.pt/bitstream/10216/57684/2/Texto%20integral.pdf>. Acesso em: 10 jun. 2019.

SOSNOSKI, D. **Entendendo e Modelando o WSDL**. Disponível em: <https://www.ibm.com/developerworks/br/library/j-jws20/j-jws20-pdf.pdf>. Acesso em: 10 jun. 2019.

W3C. **SOAP Versão 1.2 Parte 1: Estrutura de Mensagens**. Disponível em: <https://www.w3.org/TR/soap12>. Acesso em: 10 jun. 2019.