Nomes: Mateus Ragazzi Balbino 2018.1907.102-2
Gabriel Costa 2017.1907.066-0

Relatório - Web Service

Com um mundo cada vez mais conectado, a necessidade de compartilhar um dado ou conhecimento em diversas plataformas de tecnologia se faz fundamental para que sistemas troquem informações, agregando seu conteúdo e modularizando seus componentes. Tecnologias como *Web Services* se popularizaram, disponibilizando acesso à diversos dados de domínios diferentes que utilizam, em geral, uma estrutura única (RESTful) de comunicação e integração para outros sistemas.

Diversos sites na Internet disponibilizam uma biblioteca destes serviços, como Any API e APIs.io, onde é possível consultar à serviços de streaming de música (Spotify), estatísticas da NBA, dicionário Oxford, entre outros. Nesse sentido, é fundamental que este conceito seja visto e praticado na graduação para completa formação do estudante.

O trabalho em questão consiste no desenvolvimento de um Web service que fornecerá consultas à uma base de filmes e atores, conforme desejo do cliente que está utilizando o serviço. Para tal, utiliza-se da estrutura RESTful para padronização e os métodos definidos no protocolo HTTP para interação com o serviço. Os dados oferecidos serão:

1. Retornar a lista de todos os atores
2. Retornar a lista de todos os filmes
3. Retornar um ator específico
4. Retornar um filme específico
5. Criar um novo filme
6. Remover um filme
7. Atualizar um filme
8. Retornar uma lista de todos atores participando em um determinado filme
9. Busca livre sobre atores e filmes

Durante o uso, o Web service pode retornar a resposta em dois formatos solicitados: XML e JSON. A escolha do cliente deve ser informada no header HTTP content-type, conforme exemplo:

`Content-Type: application/xml`	`Content-Type: application/json`
Exemplo de consulta para retorno em XML.	Exemplo de consulta para retorno em JSON.

Para implementação, utilizamos o servidor HTTP implementado no primeiro trabalho para executar o Web service a ser desenvolvido. Para o funcionamento correto, precisamos realizar ajustes no servidor para se adequar ao padrão RESTful e receber requisições do tipo POST/PUT. Ajustamos a função de parseRequest, dentro de Http.java para leitura dos dados de interesse (Body da Requisição ou Query Params), conforme o método de requisição informado pelo cliente (GET, POST, PUT ou DELETE).

Outros ajustes executados foram: A transformação da classe ResponseFactory que criava diferentes tipos de resposta (DownloadResponse, CgiResponse, HMLResponse) para se tornar uma espécie de *Router*, que vai saber retornar o recurso solicitado pelo cliente conforme o caminho passado na URL, assim como o método HTTP utilizado e se há parâmetros na URL do recurso desejado. Nesta classe há também 2 validações: uma que vai validar se o caminho passado existe, retornando um 404

caso não exista e também outra validação do método HTTP utilizado, retornando um erro 405 caso não seja um método HTTP válido.

Com o servidor ajustado, partimos para a implementação do Web Service. Mantemos o Java como linguagem para facilitar a comunicação como servidor, onde criamos uma estrutura que separa código de banco de dados do código de aplicação/rest. No contexto de banco de dados utilizamos uma ConnectionFactory para ser a responsável pelo gerenciamento de conexões com o banco de dados, além disso criamos DAO's(Data Access Objects) que representam as tabelas utilizadas (Actors, Movies) e Entities para representação e mapeamento dos objetos de interesse. Na camada de *rest*, implementamos Controllers para se ter código responsável por receber a requisição do usuário e atendê-la, utilizando-se de DAOs para acesso aos dados e objetos que representam *Request* e *Response* do HTTP. A classe *Response* possui todos atributos necessários para se construir uma resposta HTTP do webservice além de um override de *toString()* que sabe retornar uma string com o HTTP montado de fato, que é utilizado la na frente quando o código da classe *HTTP* executa o *OutputStream* para o socket cliente.

Desenvolvemos também testes de unidade para facilitar nosso desenvolvimento e validar os retornos dos métodos responsáveis pela execução dos *endpoints*, sendo fundamental para o projeto visto que conseguimos ganhamos agilidade na implementação das DAO's e posteriormente nos serviu como meio de certificação que o sistema estava íntegro mesmo após ajustes/novas implementações.

Para facilitar a implementação, utilizamos o Maven como gerenciador de dependências do projeto para integração com bibliotecas externas(JUnit, mysql-driver, JDBC Driver, GSON (biblioteca para uso de JSON dentro do Java), entre outras dependências do projeto.

Por fim, colocamos o serviço dentro de 2 containers:

- MySQL, executando na porta padrão 3306;
- Openjdk-8 + Maven pré instalado;

O container do MySQL ficou responsável por disponibilizar o banco e populá-lo na 1º vez através do SQL disponibilizado no projeto. Já o container da aplicação do servidor, que utilizou a imagem disponibilizada pelo Miguel Doctor, retirou a necessidade da declaração de um *dockerfile* que realizasse a instalação e configuração do Maven, possibilitando o uso apenas do *docker-compose.yml* com os comandos necessários.

Vale notar que a inicialização do container com o Maven é consideravelmente demorada, pois é necessário instalar inúmeras dependências para que se possa utilizar o Maven na imagem do repositório.

Este trabalho foi bastante desafiador, visto que apesar de os integrante possuírem certa experiência com serviços RESTful, estudarem e trabalharem profissionalmente com Java e terem um trabalho com as bases prontas para o servidor HTTP, implementar um webservice RESTful "na mão", que tenha uma boa estrutura orientada à objetos traz diversas dificuldades que as frameworks abstraem com maestria.