PSEUDOREST: O REST como ele não é!

Lucas Gontijo de Souza Rodrigues

[1]Ciência da Computação
Centro Universitário do Triângulo (UNITRI) Uberlândia – MG – Brasil

lucasgontijo0@gmail.com

Abstract. *The purpose of this study is to perform an analysis on a discussion of REST creator Roy Thomas Fielding, where he exposes a critique of implementations that claim to use the architectural model proposed in his thesis. Implementations of APIs that manifest themselves as RESTFUL are used as object of study. Architectural Constraints are used which refer to important features of the model to verify these implementations, where they must succeed in these comparisons with the constraints, so that they can prove to implement the model. The conclusion is that there are in fact several implementations that are improperly named, where they have several similarities to the model in their implementation, but do not agree with the constraints that must be implemented so that it can be considered an instance of the model.*

Resumo. *O objetivo deste estudo é realizar uma análise sobre uma discussão do criador do REST Roy Thomas Fielding, onde ele expõe uma crítica sobre implementações que afirmam usar o modelo arquitetural proposto em sua tese. As implementações de APIs que se manifestam como RESTFUL são usadas como objeto de estudo. São usadas as "Architectural Constraints" que se referem a características importantes do modelo para verificar essas implementações, onde elas devem ter sucesso nessas comparações com as restrições, para que possam provar implementar o modelo. É chegada a conclusão que, de fato, existem várias implementações que são nomeadas de forma inadequada, onde têm várias semelhanças com o modelo em sua implementação, mas não estão de acordo com as restrições que devem ser implementadas para que possa ser considerada uma instância do modelo.*

1. Introdução

Existe uma confusão muito grande sobre os conceitos do estilo arquitetural REST (*Representational State Transfer*), onde diversas implementações são nomeadas como tal incorretamente. O criador do REST, Roy Fieldman, tem feito diversas críticas acerca de diversas implementações que somente se assemelham ao seu modelo arquitetural, mas que não são de fato uma instância do mesmo, logo essas implementações se nomeiam como REST de forma errada.

O objetivo deste estudo é desmistificar toda essa discussão que vem se arrastando por alguns anos onde são feitas implementações que se denominam REST mas que por algumas séries de motivos não são. A análise de tais implementações serão feitas pelas chamadas *"Architectural Constraints"*, onde são referenciadas características importantes do modelo REST. Onde cada API (*Application Programming Interface*) deverá obter sucesso nestas restrições para que possa se declarar como RESTFUL.

As seções serão divididas de tal forma que na Seção 1, os conceitos fundamentais da arquitetura REST serão abordados, para que o leitor tenha algum embasamento teórico para entender as comparações propostas pelo mesmo; a Seção 2 abordará algumas APIs, onde será feito a análise do funcionamento destas, com exemplos sobre o uso as abordagens necessárias para a sua utilização; na Seção 3, será feita a análise, utilizando as *constraints* definidas por Roy Fielding em sua tese e também a abordagem que o mesmo propõe em seu blog pessoal, a fim de provar se as implementações denominadas REST são de fato o que dizem ser; por fim, na Seção 5, serão feitas as conclusões e serão apresentadas as propostas sobre trabalhos futuros.

2. REST

O intuito da criação do REST foi a formalização das melhores práticas para o funcionamento de sistemas distribuídos e uma dessas implementações nós conhecemos como a Web. O REST possui alguns conceitos interessantes que serão explanados nas próximas subseções.

2.1. Semântica de Recursos

Todo serviço REST é baseado nos chamados recursos, que são entidades definidas no sistema, possuem identificadores e endereços próprios. Essas entidades devem possuir uma URI própria e que deve ser significativa como por exemplo: supondo que esteja sendo desenvolvida uma aplicação para algo relacionado à alunos, poderíamos utilizar o identificador "/alunos". De acordo com o modelo, essa URI realizará a interação com todos os alunos do sistema. Para tratar de alunos específicos, são usados identificadores, por exemplo, poderia ser feito a busca do aluno com a seguinte URI "/alunos/1", não foi utilizado o nome do aluno pois poderiam haver vários alunos com o mesmo nome, porém o código do aluno dentro do sistema é único.

2.2. Interação por métodos

Para interagir com as URLs, os métodos HTTP são utilizados e são responsáveis por provocar alterações nos recursos identificados pelas URIs.

Essas modificações foram padronizadas de tal maneira que:

- GET - Recupera os dados do recurso.
- POST - Cria um novo recurso.
- PUT - Atualiza um recurso.
- PATCH - Atualiza atributos específicos do recurso.
- DELETE - Apaga um recurso.

Essas chamadas HTTP possuem retornos:

- 1xx Informativo
- 2xx Sucesso
- 3xx Redirecionamento
- 4xx Erro de cliente
- 5xx Erro de servidor

2.3. Representações Distintas

Um dos pilares do REST é o uso de *media types* para alterar as representações de um mesmo conteúdo. Utilizando o exemplo anterior dos alunos, seria adequado fazer a seguinte pergunta: "Eu preciso de uma foto do aluno ou da descrição dele?". Tudo isso pode ser feito utilizando a mesma URL, especificando nos *headers* qual o formato desejado no retorno da requisição. Exemplo de especificação de retorno: "accept":"application/json" e "content-type":"application/json".

2.4. Uso correto de status code

REST também preza pelo uso correto dos status code HTTP, prezando por um retorno coerente com a operação realizada no recurso. Como exemplo, durante a criação de um recurso, seria interessante o retorno "201, Created" e o cabeçalho *Location*, indicando a localização do recurso criado, ou então em uma situação onde duas pessoas tentam se registrar com as mesmas informações, uma delas deverá receber o status "409, Conflict" com o cabeçalho *Location*, indicando onde está o recurso que originou o conflito.

2.5. Architectural Constraints

REST possui algumas especificações arquiteturais chamadas de *"Architectural Constraints"*, são divididas em 6 grupos sendo um deles opcional, cada um referenciando as características mais importantes do modelo REST.

Cliente-Servidor (Client-Server): Exige que um sistema RESTful siga o modelo Cliente-Servidor. Nesse modelo as responsabilidades de cada uma das partes ficam claramente divididas. Deixando todo ou grande parte do código do lado do servidor, que expõe essas funcionalidades através de uma interface uniforme ao cliente (Web-Service) e, consequentemente, minimiza problemas na aplicação provenientes de código no cliente.

Sem Estado (Stateless): Serviços RESTful devem ser *stateless*, ou seja, não devem guardar estado, isso explicita que, dada uma requisição, todas as informações para o processamento desta devem estar presentes no *payload* enviado ao serviço.

Cache: Uma necessidade importante, é essencial que os serviços REST sejam cacheados, ou seja, o cliente deve ser capaz de acessar um cache das respostas, evitando que sejam feitas requisições seguidas a recursos que não sofreram alterações, o cache é realizado baseado no conceito de idempotência (efeito que uma mesma requisição tem do lado do servidor. Se a mesma requisição, realizada múltiplas vezes, provoca alterações no lado do servidor como se fosse uma única, então esta é considerada idem-potente.) e permitindo, portanto, um menor consumo de banda. Além disso, quando um desses recursos for atualizado, esse cache deve ser capaz de automaticamente notar isso e responder a requisição atualizada ao cliente.

Interface / Uniform Contract: Um sistema RESTful deve ser capaz de prover uma interface uniforme. O REST usa o conceito de interface (contrato entre o cliente e o serviço) onde é necessário utilizar padrões. Isso é necessário pois caso seja preciso utilizar uma rede de serviços REST, os padrões aplicados farão as implementações entender uns aos outros.

- **Identificação de recursos**: é usado o padrão URI para identificar um recurso. Neste caso um recurso é um documento na Web.

- **Manipulação de recursos**: As URIs são usadas para manipular um recurso. A interação entre o cliente e o servidor é baseada na utilização do protocolo HTTP para descrever a comunicação. Ao se utilizar o método GET (obter) do protocolo em cima de uma URI, significa que deseja-se obter alguma informação referente ao recurso, identificado pela mesma. ”GET http://www.exemplo.com/books HTTP/1.1” de forma abstrata pode ser entendido como ”Obter uma lista de livros”. Enquanto ”GET http://www.exemplo.com/books/ISBN789 HTTP/1.1” pode ser entendido como ”Obter as informações referentes a um livro específico”.
- **Mensagens auto-descritivas**: Utiliza tipos padrões e vocabulários para tornar as mensagens auto-descritivas. Assim, o cliente pode encontrar os dados, verificando a semântica e não precisa conhecer a estrutura de dados específica que o serviço usa.
- **Hipermídia como motor de estado da aplicação (HATEOAS)**: O HATEOAS (*Hypermedia as the Engine of Application State*) define que, dentro de uma aplicação REST, o estado da aplicação e da iteração do usuário deve ser controlado através das URLs (Hypermedia) retornados pelo serviço. Logo, estando em um estado, deve se ter todos os possíveis “caminhos” que podem ser acessados a partir dali, como as URLs dos serviços que podem ser invocados no passo seguinte. Segundo [Fielding 2000], uma API REST deve ser usada sem nenhum conhecimento além da URI inicial e um conjunto de tipos de mídia padronizados. A partir desse ponto, toda transição de estado deve ser guiado pelo cliente, pelas escolhas providas pelo servidor, que estão presentes nas representações recebidas ou implícitas pela manipulação do usuário dessas representações. As transições podem ser determinadas (ou limitadas) pelo conhecimento do cliente dos tipos de mídia e mecanismo de comunicação de recursos.
- **Sistema em Camadas (*Layered-System*)**: Outro ponto interessante para o cliente é que os serviços REST devem poder ser separados em camadas de forma transparente ao usuário. Isso implica que, se quisermos colocar um serviço intermediário ou um *load balancer* entre o serviço final e o consumidor desse sistema, não deve haver grandes alterações em ambos os lados, sendo essa transição o mais transparente possível.

3. APIS

3.1. OMDb API (*The Open Movie Database*)

Segundo o site OMDb[1], a OMDb API [Fritz 2013] é um serviço da Web RESTful para obter informações de filmes, onde todos os conteúdos e imagens são contribuídos e mantidos pelos usuários. A Figura 1 mostra a página inicial da OMDb API.

Para o uso da API é necessária uma chave que pode ser obtida através do site do serviço, o serviço pode ser chamado através da URL `http://www.omdbapi.com/?apikey=[yourkey]&` conforme exemplo abaixo, e que será a nossa URL raiz.

[1]OMBb: http://www.omdbapi.com

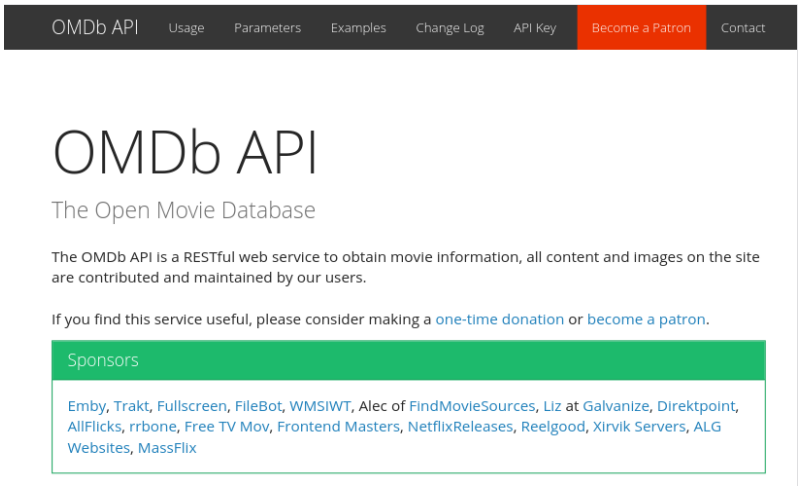

Figure 1. Página inicial da OMDB API.

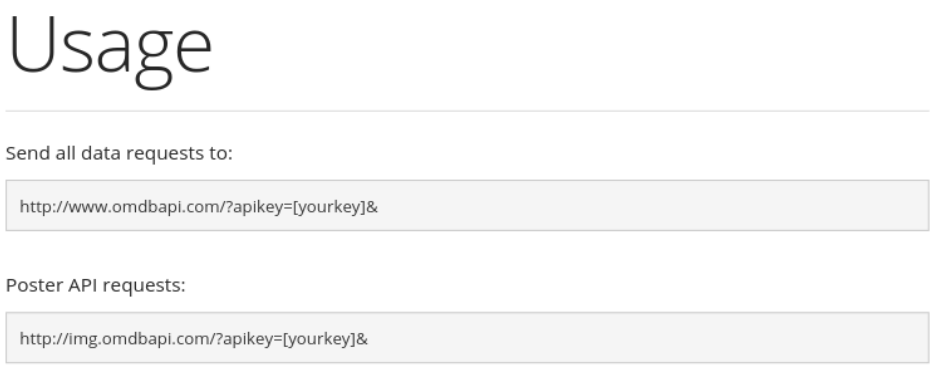

Figure 2. Exemplo de uso, Documentação OMDB API.

Realizando a chamada da API em sua URL base, obtivemos o seguinte retorno.

```
GET /?apikey=36ce0e14 HTTP/1.1
HTTP/1.1 200 OK
{
    "Response": "False",
    "Error": "Something went wrong."
}
```

Posteriormente em sua documentação, são exibidos os parâmetros para a busca com a API, que são *query parameters* assim como a chave da API.

Parameters

By ID or Title

Parameter	Required	Valid Options	Default Value	Description
i	Optional*		<empty>	A valid IMDb ID (e.g. tt1285016)
t	Optional*		<empty>	Movie title to search for.
type	No	movie, series, episode	<empty>	Type of result to return.
y	No		<empty>	Year of release.
plot	No	short, full	short	Return short or full plot.
r	No	json, xml	json	The data type to return.
callback	No		<empty>	JSONP callback name.
v	No		1	API version (reserved for future use).

*Please note while both "i" and "t" are optional at least one argument is required.

Figure 3. Parâmetros por ID e Título, Documentação OMDB API.

By Search

Parameter	Required	Valid options	Default Value	Description
s	Yes		<empty>	Movie title to search for.
type	No	movie, series, episode	<empty>	Type of result to return.
y	No		<empty>	Year of release.
r	No	json, xml	json	The data type to return.
page New!	No	1-100	1	Page number to return.
callback	No		<empty>	JSONP callback name.
v	No		1	API version (reserved for future use).

Figure 4. Parâmetros por Busca, Documentação OMDB API.

Existem diversos parâmetros dentro da API, sendo importante observar os principais, onde é possível buscar um filme pelo ID, Título e pelo tipo ao qual ele pertence (Série, Filme...), apesar do OMDB especificar em sua página que não há um vínculo direto com o IMDB, são utilizados os IDS presentes no IMDB como referência para se acessar os conteúdos da API.

Para contextualizar, será utilizado um filme existente no IMDB que pode ser encontrado no endereço: `http://www.imdb.com/title/tt0974015/`


```
{
    "Title": "Justice League",
    "Year": "2017",
    "Rated": "PG-13",
```

```
    "Released": "17 Nov 2017",
    "Runtime": "121 min",
    "Genre": "Action, Adventure, Fantasy",
    "Director": "Zack Snyder",
     ...
    "Language": "English",
    "Country": "USA",
    "imdbID": "tt0974015",
    "Type": "movie",
}
```


Requisição utilizando o parâmetro “s”, é responsável por fazer a busca utilizando o título do filme.


```
{
 "Search": [
     {
         "Title": "The Lord of the Rings",
         "Year": "1978",
         "imdbID": "tt0077869",
         "Type": "movie",
         "Poster": "https://...com/images/M/...@@._V1_SX300.jpg"
     },
     ...
     {
         "Title": "The Lord of the Rings: The Third Age",
         "Year": "2004",
         "imdbID": "tt0415947",
         "Type": "game",
         "Poster": "http://...imdb.com/images/M/...@@._V1_SX300.jpg"
     }
 ],
 "totalResults": "43",
 "Response": "True"
}
```


É importante observar que utilizando os parâmetros de busca, o valor “*page*” assume o valor 1 como *default*, logo é possível assumir que os resultados retornados são paginados.

3.2. REST Countries

Segundo o site REST Countries [RESTCountries 2017], a API REST Countries provem informações sobre países utilizando uma interface RESTFUL, A Figura 5 mostra a página inicial da API.

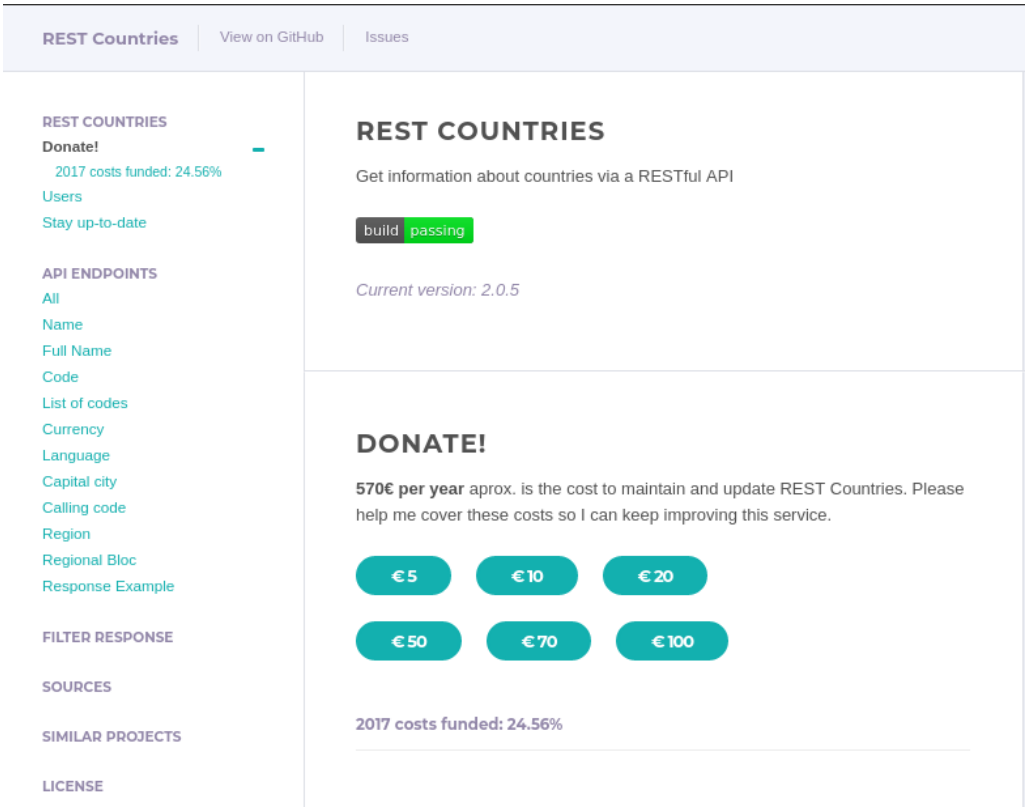

Figure 5. Página inicial da REST Countries API.

Para a utilização da API não é necessário qualquer tipo de autenticação, portanto pode ser realizada a chamada conforme exemplo abaixo.

Chamada realizada na URL raiz:

```
GET /v2 HTTP/1.1
HTTP/1.1 200 OK
[
    {
        "name": "Afghanistan",
        "topLevelDomain": [".af"],
        "alpha2Code": "AF",
        "alpha3Code": "AFG",
        "callingCodes": ["93"],
        "capital": "Kabul",
        "region": "Asia",
        "subregion": "Southern Asia",
        "population": 27657145,
        "latlng": [33,65],
        "demonym": "Afghan",
        "area": 652230,
        "gini": 27.8,
        "timezones": ["UTC+04:30"],
```

```
        "borders": ["IRN","PAK","TKM","UZB","TJK","CHN"],
        "numericCode": "004",
        ..,
        "flag": "https://restcountries.eu/data/afg.svg",
        ...
        "cioc": "AFG"
    },
    ...
    {
        "name": "Zimbabwe",
        "topLevelDomain": [".zw"],
        "alpha2Code": "ZW",
        "alpha3Code": "ZWE",
        "callingCodes": ["263"],
        "capital": "Harare",
        "altSpellings": ["ZW","Republic of Zimbabwe"],
        "region": "Africa",
        "subregion": "Eastern Africa",
        "population": 14240168,
        "latlng": [-20,30],
        "demonym": "Zimbabwean",
        "area": 390757,
        "gini": null,
        "timezones": ["UTC+02:00"],
        "borders": ["BWA","MOZ","ZAF","ZMB"],
        "nativeName": "Zimbabwe",
        "numericCode": "716",
        ...
        "flag": "https://restcountries.eu/data/zwe.svg",
        ...
        "cioc": "ZIM"
    }
]
```


O retorno possui diversos países e suas informações, a grande maioria foi omitida devido ao tamanho do *payload* de resposta, aproximadamente 19 mil linhas. Posteriormente em sua documentação, são exibidos os caminhos dos recursos para se realizar buscas com a API, onde pode-se ter os seguintes critérios de busca:

Name: Pesquisa por nome do país, pode ser o nome nativo ou o nome parcial.
Exemplo: `https://restcountries.eu/rest/v2/name/united`

Full Name: Pesquisa país pelo nome completo.
Exemplo: `https://restcountries.eu/rest/v2/name/aruba?fullText=true`

Code: Pesquisa pelo código ISO 3166-1. Código de país com 2 ou 3 letras.
Exemplos:
`https://restcountries.eu/rest/v2/alpha/co`

`https://restcountries.eu/rest/v2/alpha/col`

Lista de códigos: Pesquisa pelo código ISO 3166-1, utilizando uma lista de parâmetros.
Exemplo: `https://restcountries.eu/rest/v2/alpha?codes=col;no`

Moeda: Pesquisa por código de moeda ISO 4217.
Exemplo: `https://restcountries.eu/rest/v2/currency/BRL`

Língua: Pesquisa pelo código de idioma ISO 639-1.
Exemplo: `https://restcountries.eu/rest/v2/lang/es`

Capital: Pesquisa por cidade capital.
Exemplo: `https://restcountries.eu/rest/v2/capital/brasilia`

Código de chamada: Pesquisar por código de chamada.
Exemplo: `https://restcountries.eu/rest/v2/callingcode/55`

Região: Pesquisa por região: África, Américas, Ásia, Europa, Oceania.
Exemplo: `https://restcountries.eu/rest/v2/region/europe`

Bloco regional: Pesquisa por bloco regional:

- EU (*European Union*)
- EFTA (*European Free Trade Association*)
- CARICOM (*Caribbean Community*)
- PA (*Pacific Alliance*)
- AU (*African Union*)
- USAN (*Union of South American Nations*)
- EEU (*Eurasian Economic Union*)
- AL (*Arab League*)
- ASEAN (*Association of Southeast Asian Nations*)
- CAIS (*Central American Integration System*)
- CEFTA (*Central European Free Trade Agreement*)
- NAFTA (*North American Free Trade Agreement*)
- SAARC (*South Asian Association for Regional Cooperation*)

Exemplo: `https://restcountries.eu/rest/v2/regionalbloc/eu`

Ao se realizar uma requisição "inválida" é retornado a seguinte resposta:

```
GET /v2/name/invalidrequest HTTP/1.1
HTTP/1.1 404 NOT FOUND
{
    "status": 404,
    "message": "Not Found"
}
```

Segundo a [RESTCountries 2017] também é possível realizar pesquisas personalizadas, onde os parâmetros que serão retornado devem estar explícitos na *query*, estes parâmetros podem ser utilizados em qualquer recurso presente na API.

```
GET
https://restcountries.eu/rest/v2/name/brazil?fields=name;capital;flag
HTTP/1.1
```

```
[
    {
        "flag": "https://restcountries.eu/data/bra.svg",
        "name": "Brazil",
        "capital": "Brasilia"
    }
]
```

4. Análise de APIs

A análise das APIs de acordo com a sua utilização apresentada na sessão anterior foi realizada utilizando as *"Architectural Constraints"* como referências para comparação. O subitem "Sistema em camadas (Layered-System)" da *constraint* "*Interface / Uniform Contract*" será omitido, pois depende de critérios como conhecimento sobre a infraestrutura para conseguir realizar algumas comprovações como, por exemplo, a existência de um *loadbalancer*, portanto, não seria possível obter de forma clara tal comprovação.

4.1. OMDB API

Realizando a análise da requisição em sua URL raiz, alguns pontos interessantes podem ser observados:

```
GET /?apikey=36ce0e14 HTTP/1.1
HTTP/1.1 200 OK
{
    "Response": "False",
    "Error": "Something went wrong."
}
```

O primeiro ponto de observação está relacionado ao retorno de sua URL raiz, onde a constraint "Interface / Uniform Contract" é violada no subitem "Hipermídia como motor de estado da aplicacão (HATEOAS)". Como citado anteriormente no capítulo referente aos conceitos de REST [Fielding 2000], uma API REST deve ser usada sem nenhum conhecimento além da URI inicial e um conjunto de tipos de mídia padronizados. A API nos fornece em sua documentação na sessão de parâmetros, um referente ao tipo de retorno de dados, onde podem ser retornados "JSON " ou "XML". Porém, somente com estas informações fornecidas não são retornados URLs (*Hypermedia*) pelo serviço. Logo, estando em um estado no caso o estado de sua URL raiz, não é possível ver todos os possíveis "caminhos" que podem ser acessados à partir dali conforme pode ser observado na requisição abaixo. Um outro ponto interessante está relacionado ao uso incorreto dos *status codes*. Durante a chamada, obtivemos sucesso mesmo com uma requisição que aparante ser "inválida", foi retornado o HTTP *status code* 200 OK, normalmente utilizado quando a requisição obtém sucesso.

Um outro ponto a ser observado é que, devido a ausência do conceito de HATEOAS, não é possível "caminhar" pela API de forma interativa, sendo necessário a consulta constante a documentação. Também é necessário que ressaltar que é necessário acesso ao site de um terceiro onde os IDs de alguns recursos são associados para realizar consultas tais consultas.

Outro ponto a ser observado é que devido a falta do conceito de HATEOAS, o uso da API fica obscuro em alguns aspectos como, por exemplo, na requisição abaixo com o objetivo de buscar um filme por título:

```
GET /?apikey=36ce0e14&s=batman HTTP/1.1
HTTP/1.1 200 OK Time: 350ms
{
    "Search": [
        {
            "Title": "Batman Begins",
            "Year": "2005",
            "imdbID": "tt0372784",
            "Type": "movie",
            "Poster": "https:..._V1_SX300.jpg"
        },

        ...
    ],
    "totalResults": "336",
    "Response": "True"
}
```

São retornado diversos títulos, porém não são retornadas URLs que indiquem os outros possíveis estados da aplicação. No caso, as possíveis páginas da consulta, sendo necessário a consulta na documentação para ter conhecimento de um outro parâmetro "*page*" para que seja possível obter os outros títulos.

```
GET /?apikey=36ce0e14&s=batman&page=2 HTTP/1.1
HTTP/1.1 200 OK Time: 407ms
{
    "Search": [
        {
            "Title": "Batman: The Killing Joke",
            "Year": "2016",
            "imdbID": "tt4853102",
            "Type": "movie",
            "Poster": "https://...images-amazon.com/images/M/...@@._V1_
        },
        ...
    ],
    "totalResults": "336",
    "Response": "True"
}
```

Outro fator importante a ser observado é que a chamada de recurso idempotente são cacheados, evitando que sejam feitas requisições seguidas a recursos que não sofreram alterações, permitindo, portanto, um menor consumo de banda.

```
GET /?apikey=36ce0e14&s=batman HTTP/1.1
```

```
HTTP/1.1 200 OK Time: 407ms

GET /?apikey=36ce0e14&s=batman HTTP/1.1
HTTP/1.1 200 OK Time: 39ms
```

É possível concluir que esta implementação não é de fato uma instância do estilo arquitetural REST, devido a não implementação das *constraints*. [Fielding 2008] ressalta em seu blog pessoal que uma interface baseada somente em HTTP não é REST e que esta abordagem está incorreta. HATEOAS é uma das *constraints* e que, se o motor de estado da aplicação não é orientado por Hipertexto, então ela não é RESTFUL, logo não é uma API REST.

4.2. REST Countries API

Existem diversos pontos a serem observado, dentre eles um importante é o retorno da URL raiz, onde a *contraint* "Interface / Uniform Contract" é violada no subitem "Hipermídia como motor de estado da aplicacão (HATEOAS)". [Fielding 2000] diz que uma API REST deve ser utilizada sem nenhum conhecimento alem de sua URI inicial e e um conjunto de tipos de mídia padronizados. A API nos fornece em sua documentação qual são os possíveis estados para se realizar a interação com a mesma, porém sem a presença da documentação não é possível navegar de maneira interativa sendo extremamente necessário a utilização da documentação. Um outro ponto interessante se dá ao fato de que a semântica de recursos não condiz com o apresentado em REST, pois para cada tipo de pesquisa existe uma recurso mapeado, onde a abordagem mais coerente seria a utilização de um recurso definido para representar a abstração dos países, como por exemplo: "/countries". Uma outra observação importante a ser feita é que esta implementação não faz o uso correto dos *media-types*. Portanto, não são implementadas as Representações Distintas, onde ao se realizar uma consulta em um recurso é retornado um link para o acesso da imagem referente a pesquisa, porém não é possível realizar a busca de uma imagem diretamente pela requisição do recurso especificando um retorno, no caso a imagem do país pesquisado.

Outro ponto interessante se dá ao fato de que por não implementar HATEOAS, ao se fazer uma requisição inválida, não são retornados os possíveis estados da aplicação, onde novamente não são implementados os conceitos de *HATEOAS*.

```
GET /v2/name/invalidrequest HTTP/1.1
HTTP/1.1 404 NOT FOUND
{
    "status": 404,
    "message": "Not Found"
}
```

É possível concluir que esta outra implementação não é uma instância de REST, devido a não implementação de diversas *contraints* e principalmente ao fato de não utilizar corretamente a semântica de recursos. A abstração de um recurso é representada por diversas entidades e não é possível realizar o uso de *media-types*. Conforme [Fielding 2008], uma implementação baseada em HTTP, mas que não implementa as *contraints* não pode ser considerada como RESTFUL.

5. Conclusão

Levando-se em conta o que foi observado, conclui-se que, de fato, existem diversas implementações que se assemelham a REST porém somente em alguns pontos de vista como modelo cliente/servidor e a utilização do protocolo HTTP para expor uma interface ao seu usuário. Porém, estas se nomeiam de forma inadequada onde na grande maioria das vezes o mais adequado seria a utilização do termo RPC (um modelo cliente/servidor, onde é entregue uma abstração de chamada de procedimento, onde é enviada ao servidor remoto para a execução de um procedimento específico e que uma resposta é retornada ao cliente).

Para possíveis trabalhos futuros, seria proposto um estudo cujo o objetivo seria analisar o porquê estas implementações são nomeadas de forma inadequada, e também seriam analisados em quais cenários seriam interessantes a implementação de um modelo baseando-se em REST e se realmente existe a necessidade de uma implementação se nomear como tal.

References

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

Fielding, R. T. (2008). Rest apis must be hypertext-driven. Disponível em `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`. Acessado em Outubro de 2017.

Fritz, B. (2013). Omdb api the open movie database. Disponível em `http://www.omdbapi.com`. Acessado em Outubro de 2017.

RESTCountries (2017). Get information about countries via a restful api. Disponível em `https://restcountries.eu`. Acessado em Outubro de 2017.