Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Um mapeamento de práticas em projetos de APIs REST

José André Carneiro Neto

Trabalho de Graduação

Recife
2020

Universidade Federal de Pernambuco
Centro de Informática

José André Carneiro Neto

Um mapeamento de práticas em projetos de APIs REST

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Fernando Castor*

Recife
2020

Cause I've seen blue skies through the tears in my eyes
And I realize, I'm going home
—DR. FRANK N. FURTER

Resumo

A web de hoje se apresenta na forma de uma coleção de aplicações interconectadas produzindo e consumindo dados. Dentro desse contexto, *web services*, aplicações focadas na comunicação automatizada entre dois sistemas, são cada vez mais prevalentes. Dentre os modelos arquiteturais propostos para a implementação de web services um se destacou por sua simplicidade e flexibilidade, o REST.

APIs REST, como são chamadas as interfaces que implementam os princípios, propriedades e restrições descritos pelo REST, se tornaram sinônimo de *web service* na última década. Como expressão de maturidade tecnológica, práticas se consolidaram como padrões a serem seguidos e foram documentadas para gestão do conhecimento.

Afim de explorar o grau de consolidação das práticas recomendadas para projetos de APIs REST, este trabalho propõem um mapeamento de práticas a partir de dois tipos de fonte: a literatura branca (documentos que passaram pelo processo editorial) e a literatura cinzenta (documentos informais não revisados por pares).

Palavras-chave: REST, web services, mapeamento, boas práticas, literatura cinzenta

Abstract

Nowadays the web presents itself as a collection of interconnected applications constantly producing and consuming data. Within this context, web services, applications focused on automated communication between two systems, are increasingly prevalent. Among the major architectural models proposed for web service specification, REST (Representational State Transfer) stood out as an example of simplicity and flexibility and REST APIs, interfaces that implement principles, properties and restrictions described by REST, have become synonymous with web service.

As a signal of technological maturity, practices are consolidated as standards to be followed and are documented for knowledge management. To explore the degree of consolidation of recommended practices for REST API projects, this work proposes a mapping of practices from two types of sources: white literature (documents that have gone through some editorial process) and grey literature (non peer-reviewed informal documents).

Keywords: REST, web services, mapping, best practices, grey literature

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Mais de três décadas após sua conceitualização, a web continua em constante crescimento [26]. Uma das maneiras como tal crescimento se evidencia é por meios da escala e quantidade de suas aplicações. A exemplo do Google e Facebook que hoje passam da marca de bilhões de usuários ativos [29][15]. Desde o início dos anos 2000, no entanto, as aplicações web não atendem apenas usuários finais, mas também são integradas com outros sistemas por meio web services [64].

Um *web service* pode ser definido como um *software* com capacidade de prover funcionalidade para outras máquinas baseadas em rede. Isto é, por meio de mensagens definidas para uma interface é possível executar remotamente procedimentos sem que seja necessário conhecer detalhes de sua implementação [43]. A fim de atingir um alto grau de interoperabilidade, o formato de mensagem e das interfaces empregadas, também conhecidas como APIs, foram formalizadas em padrões arquiteturais. Dentre os quais destaca-se o REST.

REST (acrônimo para Representational State Transfer) é um modelo arquitetural criado em 2000 por Roy Fielding como forma de especificar um conjunto de princípios, propriedades e restrições para um necessárias para projeto de aplicação web. Veterano dos padrões web, Fielding utilizou sua experiência na especificação do HTTP para imaginar uma web como uma rede de máquinas de estado, onde o usuário navega a partir de links (transições) e tem acesso à diferentes recursos (estados) [1].

Fortemente embasado em tecnologias que já faziam parte da web, o REST logo se popularizou e suas especificações se tornaram sinônimo de web service. O modelo proposto em 2000, no entanto, não funcionava como estilo universal para implementação de serviços na web, sendo algumas de suas restrições adaptadas para atender às mais diferentes realidades. Como por exemplo a restrição *Code-On-Demand*, sem tanta adoção no âmbito de serviços web. Fato este já previsto por Roy Fielding na especificação original do REST [24].

Dentro desse contexto de adaptações, diferentes práticas surgiram como extensão ao conjunto originalmente proposto e continuam em processo de evolução até os dias atuais [41]. Evolução essa baseada nas relações entre literatura e implementação prática, um relacionamento cíclico onde a primeira serve de base para a última enquanto também é influenciada e revisada a partir das experiências de mundo real. A relação direta dos dois tipos de literatura é objeto de estudo de

Garousi, que ressalta a necessidade da inclusão de literatura cinzenta nas revisões sistemáticas em engenharia de software [31] dado o seu potencial benefício em cobertura e alinhamento com o estado da prática vigente. Assim, vê-se como necessária uma revisão das práticas de projetos de APIs REST documentadas na literatura cinzenta e acadêmica, representando respectivamente uma contexto mais próximo e mais distante do praticado pelos desenvolvedores.

1.1 Motivações

Vinte anos passados de sua conceitualização, o REST tem sido considerado sinônimo de serviços web por vários anos. Em meio a tal popularidade, determinadas práticas se tornam padrão de indústria e são recomendadas por diversas fontes da literatura cinzenta, isto é, posts em blogs, sites e artigos não revisados por pares.

O cenário, no entanto, não é tão claro quando considerada a literatura acadêmica. Levando em consideração fatores como cobertura de práticas recomendadas e a distribuição das mesmas de acordo com fatores de qualidade de software, vê-se necessário um estudo para o entendimento da atual distância entre a literatura acadêmica e a literatura cinzenta no âmbito das práticas recomendadas em projetos de API REST, assim como a melhor compreensão, de maneira mais abrangente, do estado da prática de projetos de APIs REST.

1.2 Estrutura

Esta monografia é dividida em 4 capítulos, além da introdução. O capítulo 2 visa a revisão dos elementos teóricos necessários para a análise das práticas coletadas. O capítulo 3 detalha o protocolo de busca utilizado na coleta das práticas em todas as suas etapas. O capítulo 4 traz o resultados das questões proposta na pesquisa. O capítulo 5 conclui a questão central da pesquisa com base nos resultados coletados e disserta sobre possíveis trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Protocolo HTTP

Desenvolvido em 1989 por Tim Berners-Lee, o *Hypertext Transfer Protocol* é um protocolo de camada de aplicação amplamente utilizado na web para sistemas distribuídos em hipermídia [12]. O HTTP carrega como requisito o emprego de um protocolo de camada de transporte resiliente a erros e inconsistências da rede, a opção mais comum para o caso é o TCP [23].

A especificação do HTTP não se limita à definição do formato de requisição e resposta, mas também define uma série de conceitos essenciais para sua implantação e detalhados a seguir.

2.1.1 Sessão HTTP

Uma sessão HTTP compreende todo ciclo de processamento usualmente dividido em três etapas [25]:

1. Uma conexão de camada de transporte é estabelecida entre cliente e servidor.

2. O cliente envia uma requisição e aguarda resposta.

3. Após processar a requisição, o servidor envia uma resposta contendo os dados requisitados e um código de status

Neste processo estão envolvidos os dois tipos de mensagens suportados pelo protocolo: requisição e resposta.

2.1.2 Cabeçalho

Suportado por ambos tipos de mensagens, os cabeçalhos se dividem entre uso geral, uso exclusivo em requisições, uso exclusivo em respostas e cabeçalhos de entidades. Todos seguindo um formato padrão *NOME:VALOR*. Não existe uma ordem específica na qual os cabeçalhos devem ser enviados, mas é considerado boa prática enviar os cabeçalhos de uso geral antes dos outros tipos [23].

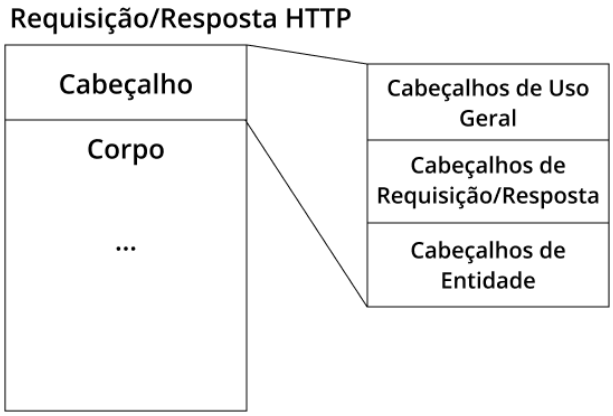


Figura 2.1 Esquema dos cabeçalhos HTTP
Fonte: Próprio autor

2.1.3 Corpo de mensagem

De presença não obrigatória, o corpo da mensagem é utilizado para envio dados associados a entidade sendo requisitada. Entende-se por entidade toda bloco bem definido de dados requisitado em uma sessão. Diretamente relacionado é o uso do cabeçalho *Content-Length* como sinalizador do tamanho em bytes da corpo de requisição/resposta [23].

2.1.4 Recurso

Gourley [35] define recurso como a fonte do conteúdo na web. Podendo variar de arquivo estático até um sistema capaz de gerar conteúdo dinamicamente. Como visto na figura 2.2, os recursos são o conteúdo central das mensagens HTTP e fazem parte da infraestrutura da web. Seu acesso é endereçado por meio de outro elemento fundamental do protocolo HTTP, a URI.

2.1.5 URI

Gourley [35] também ressalta que cada recurso, disponibilizado por um servidor web, tem um nome associado de forma que os clientes consigam acessar os recursos de interesse. Tal nome é chamado de *uniform resource identifier* (identificador uniforme do recurso) ou URI.

2.1.6 URL

A URL, sigla para *uniform resource locator* (localizador uniforme do recurso) é o subtipo mais comum de URI. Sua estrutura pode ser dividida em três partes [35]:

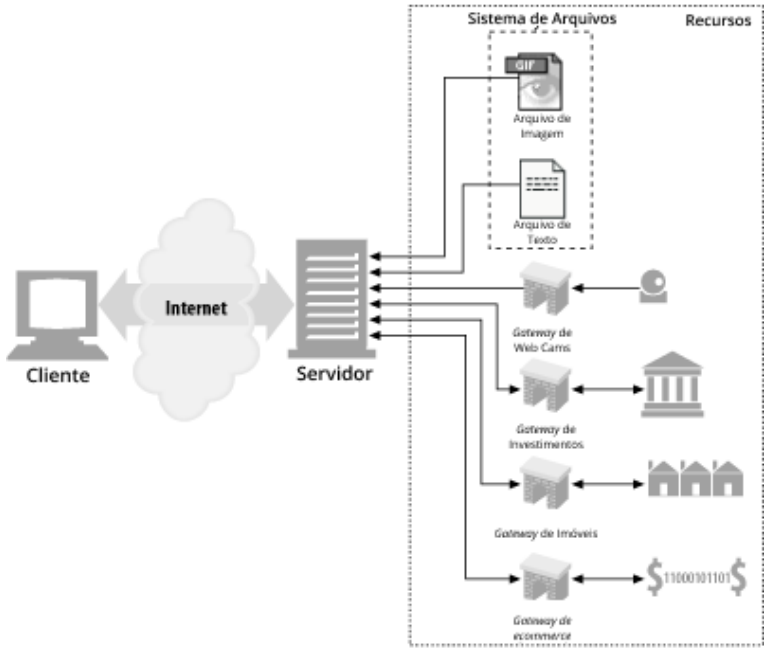


Figura 2.2 Um recurso como qualquer provedor de conteúdo web.
Fonte: Gourley, 2002

Figura 2.3 URLS especificam protocolo, servidor e recurso local
Fonte: Gourley, 2002

- A primeira se chama esquema e é responsável por descrever o protocolo utilizado para acesso ao recurso. Por exemplo (http:// ou https://).

- A segunda parte determina o nome do servidor acessado. Por exemplo (www.google.com).

- A terceira parte compreende todo o texto após o nome do servidor e identifica localmente o recurso. Por exemplo (/produtos/descricao).

2.1.7 Método HTTP

O método HTTP é um token cuja função é indicar a operação requisitada pela cliente. Os métodos especificados são *GET*, *HEAD*, *POST*, *PUT*, *DELETE*, *TRACE*, *OPTIONS*, *CONNECT* e *PATCH*. A RFC 2616 especifica comportamentos padrão como visto na tabela 2.1.

Método	Função
GET	Recuperar qualquer informação especificada pela URI
HEAD	Idêntico ao GET, porém contendo apenas o cabeçalho em sua resposta
POST	Adiciona um novo subordinado ao recurso especificado pelo URI
PUT	Solicita armazenamento da entidade na URI especificada
DELETE	Apaga o recurso especificado
TRACE	Retorna a própria requisição para fins de auditoria
OPTIONS	Retorna os métodos disponível em determinada URI
CONNECT	Utilizado para converter conexões HTTP em túneis para *hosts* remotos
PATCH	Atualiza parcialmente o recurso especificado

Tabela 2.1 Métodos HTTP e suas respectivas funções

2.1.8 Código de status

Segundo Fielding [23], o código de status é um código numérico de 3 dígitos que indica o resultado da tentativa de satisfazer a requisição. O primeiro dígito do código de status define a sua classe de resposta enquanto os últimos dois não indicam nenhuma categorização. Em termos de classe, os códigos de status se dividem da seguinte forma:

- 1xx Informativo - Requisição recebida, continuando processo
- 2xx Sucesso - A ação foi corretamente recebida, entendida e aceita
- 3xx Redirecionamento - Uma nova ação deve ser executada afim de completar a requisição

- 4xx Erro do Cliente - A requisição contém um erro de sintáxe ou não pode ser satisfeita
- 5xx Erro do Servidor - O servidor falhou em satisfazer uma requisição válida

2.2 Web Services

Para Gottschalk [34], *web service* é uma interface que descreve uma coleção de operações acessíveis pela rede através de mensagens padronizadas.

Kalin [49], por sua vez, levanta três pontos nos quais *web services* diferem de outros tipos de sistemas distribuídos:

Infrastrutura aberta
Web services utilizam protocolos e linguagens amplamente conhecidas e independentes, a exemplo do HTTP, XML e JSON.

Transparência de plataforma e linguagem
Web services são uma excelente forma de integrar serviços escritos em linguagens diferentes e mantém interoperabilidade como valor inerente.

Design modular
Web services podem ser compostos por outros *web services* já existentes. Análogos a pequenos pedaços de *software*, os serviços são desenvolvidos como operacões simples e componetizados para construção de uma funcionalidade mais complexa.

2.2.1 SOA

SOA, acrônimo para Service-Oriented Architecture (Arquitetura Orientada a Serviços), é um estilo arquitetural cuja ideia central é que uma aplicação resulta da integração de diversos serviços acessíveis em rede. Tal integração sendo possível por conta de uma interface com claras definições das operações e seus detalhes de invocação disponíveis. [49].

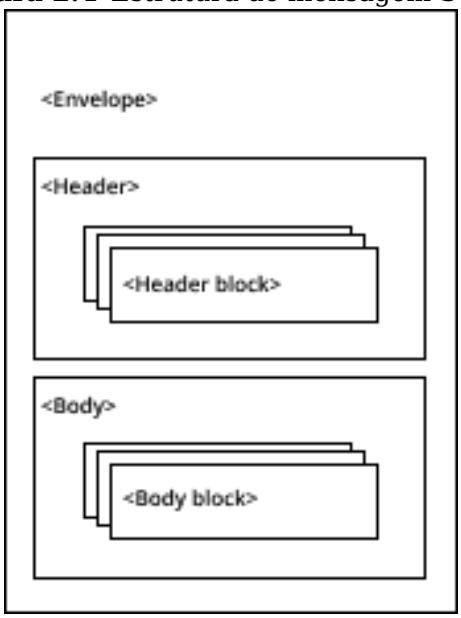
2.2.2 SOAP

SOAP, acrônimo para Simple Object Access Protocol (Protocolo Simples de Acesso a Objeto) é um protocolo de mensagem que permite aplicações remotas se comunicarem utilizando HTTP e XML como tecnologias base [37].

No uso do protocolo, mensagens são vistas como envelopes, que por sua vez são subdivididos em *<Header>* (cabeçalho) e *<Body>* (corpo). O cabeçalho é um elemento opcional e contém blocos de informação relevantes para o processamento

da mensagem. Já o corpo concentra a informação utilizada pela aplicação final e de contexto da aplicação.

Figura 2.4 Estrutura de mensagem SOAP



Fonte: Adaptado de Papazoglou, 2008

Halili também reitera [37] que o SOAP estabelece dois tipos de requisição de mensagem:

Remote Procedure Call
: Chamadas de procedimento remoto, ou RPC, como o nome sugere, são representações da execução de funções em um sistema remoto. Usualmente síncrono, é implementado no modelo cliente-servidor e tem um modelo definido de chamadas e passagem de parâmetros.

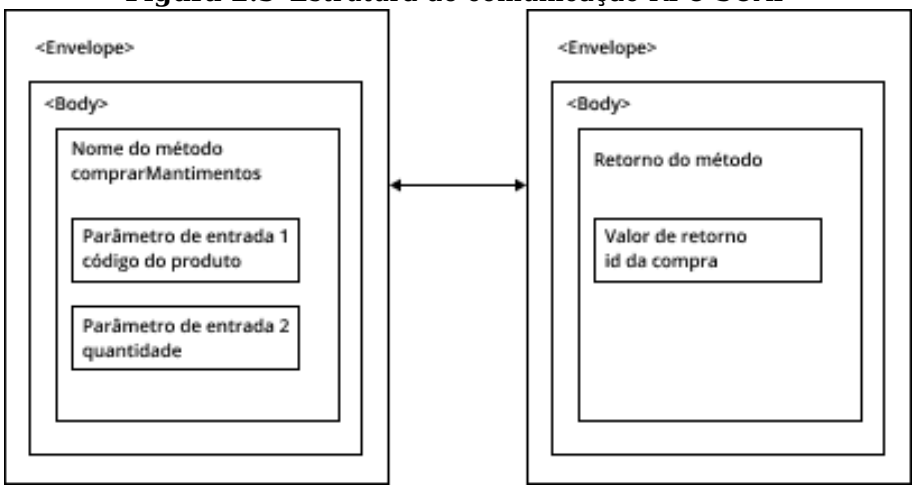
Document Requests
: O corpo do envelope é composto por um documento no formato XML sem estrutura pré-definida pelo SOAP. A conteúdo do corpo é transferido sem modificações para a aplicação, que por sua vez interpreta o documento e por sua vez retorna outro documento.

As figuras 2.7 e 2.8 exemplificam o formato de mensagem SOAP utilizando em comunicações do tipo RPC.

2.2.3 REST

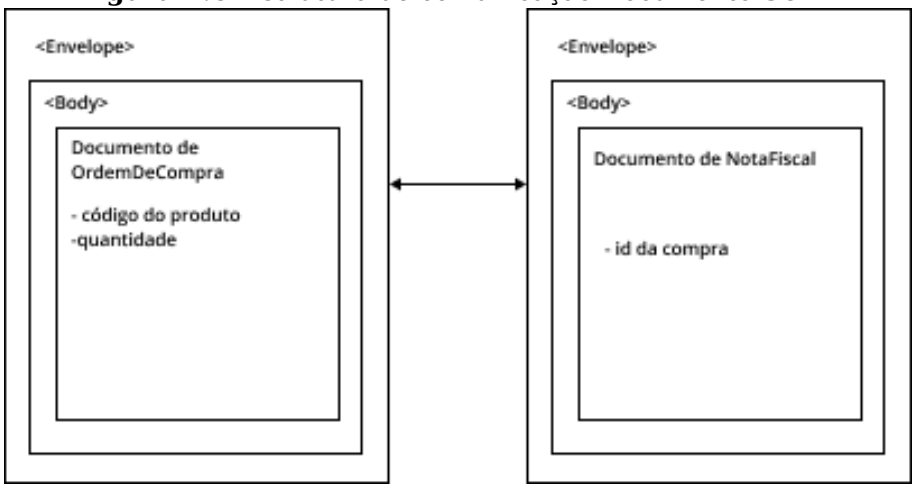
Criado por Roy Fielding em 2000, REST, acrônimo para *Representational State Transfer* (Transferência Representacional de Estado) é um estilo arquitetural para

Figura 2.5 Estrutura de comunicação RPC-SOAP



Fonte: Adaptado de Papazoglou, 2008

Figura 2.6 Estrutura de comunicação Documento-SOAP



Fonte: Adaptado de Papazoglou, 2008

aplicações remotas sobre o HTTP. Em sua tese de doutorado, Fielding elenca seis restrições para que os sistemas remotos se tornem *restful*:

2.2.3.1 Client-Server

Fortalecendo o princípio de separação de preocupações, a restrição cliente-servidor por definição trata a aplicação como duas partes interconectadas e independentes. A conformidade com a restrição também traz benefícios relacionados a portabilidade e escalabilidade, cruciais para o crescimento da internet como existe hoje.

```
POST /InStore HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/store">
  <m:GetProductPrice>
    <m:productCode>38762</m:productCode>
  </m:GetProductPrice>
</soap:Body>

</soap:Envelope>
```

Figura 2.7 Exemplo de requisição RPC-SOAP

```
  HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/store">
  <m:GetProductPriceResponse>
    <m:Price>164.1</m:Price>
  </m:GetProductPriceResponse>
</soap:Body>

</soap:Envelope>
```

Figura 2.8 Exemplo de resposta RPC-SOAP

2.2.3.2 Stateless

Aplicada sobre a restrição anterior, o *stateless* (sem estado) exige que toda requisição do cliente para o servidor contenha toda a informação necessária para seu entendimento. Dada a natureza atômica das requisições em conformidade com a restrição, propriedades como visibilidade, confiabilidade e escalabilidade são positivamente afetadas.

2.2.3.3 Cache

Aplicada sobre um cenário *stateless*, a restrição de suporte à cache determina que retornado como resposta a uma requisição devem ser explicitamente rotulados como *cacheable* ou não *cacheable*. Para os casos positivos, o cliente está então livre para reutilizar a resposta previamente recebida em requisições futuras. Com ganhos em eficiência, escalabilidade e usabilidade, o uso de cache reduz significativamente o tempo de resposta para o usuário.

2.2.3.4 Uniform Interface

Segundo Fielding [24], a principal funcionalidade que difere o REST de outros arquiteturas de sistemas baseados em rede é a sua interface uniforme entre componentes. Para garantir a uniformidade das interfaces em sua implementação sistemas *restful* seguem quatro restrições:

Identificação de recursos
: Recursos devem ser unicamente identificados por uma URI e seu formato de representação indepentende da maneira como a aplicação o armazena.

Manipulação de recursos através de representações
: Quando em posse da representação de um determinado recurso, incluindo qualquer metadado necessário, o cliente tem poder para expressar intenção de modificar ou apagar o recurso no servidor.

Mensagens auto-descritivas
: Cada mensagem é composta, além dos dados do recurso, por metadados que indicam como a mensagem deve ser processada.

Hypermedia as the Engine of Application State (HATEOAS)
: O HATEOAS, ou Hipermídia como Motor do Estado da Aplicação, é uma restrição que determina um padrão endereçamento entre recursos relacionados de forma análoga ao já utilizado no web, navegação por links.

  Enquanto páginas web possibilitam a navegação entre outros páginas apontadas, o HATEOAS determina que representações de recursos apontem para outros recursos por meio de sua URI. O emprego dessa restrição diminui o

acoplamento do cliente e servidor, além possibilitar descoberta automatizada de recursos relacionados.

2.2.3.5 Layered System

Mais voltado para melhorias na escalabilidade e modularização, é proposta a restrição de sistemas distribuídos em camadas. Tal restrição permite que o sistema seja composto por diferentes camadas, cada uma com sua função específica, e que a adição ou remoção de uma camada não cria necessidade de adaptação no formato da mensagem.

2.2.3.6 Code-On-Demand

Na restrição Code-On-Demand (código sob demanda), o REST permite a extensão de funcionalidades do cliente por meio do *download* e execução de código fornecido pelo servidor [24]. A conformidade com essa restrição é vista como opcional e pode ser exemplificada por *applets* Java e *scripts* Javascript.

2.2.4 Comparação SOAP e REST

Criado como resposta à crescente complexidade do SOAP, o REST é, tanto no formato de resposta quanto na quantidade de elementos funcionais, mais simples que o SOAP. A tabela 2.2 compara diretamente as duas abordagens.

Tabela 2.2 Comparação entre SOAP e REST

SOAP	REST
Mudanças no serviço em um cenário de provisionamento SOAP às vezes requerem alterações complicadas no cliente	Mudanças no serviço em um cenário de provisionamento REST não requerem alterações no cliente
SOAP tem um *payload* pesado quando comparável ao REST	REST é definitivamente leve, uma vez que é projetado para transferência de dados leves sobre a interface mais comum, a URI
Requer conversar de binários anexados	Suporta todos os tipos de dado diretamente
SOAP não é amigável a infraestruturas de rede sem fio	REST é amigável a infraestruturas de rede sem fio
Um *web service* SOAP sempre retorna dado no formato XML	Serviços baseados em REST provêm flexibilidade no formato do retorno
Consome mais banda disponível, uma vez que a requisição SOAP pode chegar requerir 10 vezes mais bytes que uma requisição REST	Consome menos banda disponível por conta do formato de mensagem leve
Requisições SOAP utilizam o método POST e são formadas por um XML de alta complexidade, o que dificulta o uso de cache nas respostas	APIs *restful* podem ser consumidas a partir de simples requisições GET e servidores *proxy* pode gerenciar a cache relacionada com facilidade
SOAP tem operações definididas pela aplicação e esquemas customizados	REST tem operações em conformidade com os métodos HTTP disponíveis
Agnóstico de plataforma, linguagem e protocolo de transporte	Agnóstico de plataforma e linguagem
Projetado para comunicação em ambiente de computação distribuída	Assume um modelo de comunicação ponto-a-ponto
Difícil de desenvolver, requere ferramentas específicas	Simples de desenvolver

Fonte: Adaptado de Wagh, 2012

Capítulo 3

Metodologia

Neste capítulo será apresentada a metodologia utilizada para coleta e análise da literatura, assim como o critério para consilidação das práticas identificadas.

Este trabalho diferencia-se dentre outros artigos contemplando boas práticas em projetos de API REST no uso de literatura cinzenta como fonte auxiliar para o mapeamento. O protocolo de busca foi executado de acordo com os passos e critérios propostos por Garousi para revisões multivocais da literatura [32].

3.1 Protocolo de busca

A partir da motivação e do estudo primário das práticas relacionadas à APIs REST foi elaborada a seguinte questão principal:

RQ0 Como a literatura cinzenta e a literatura branca se diferem quanto à recomendação de boas práticas em projetos de APIs REST?

Afim de auxiliar no processo de entendimento da questão principal, foram elaboradas as seguintes questões secundárias:

RQ1 *Quais as boas práticas recomendadas para projetos de APIs REST?*

RQ2 *Como a literatura cinzenta e a literatura branca se distribuem quando relacionadas práticas em projetos de APIs REST a critérios de qualidade de software?*

RQ3 *Quais as tendências na produção de artigos sobre práticas em projetos de APIs REST na literatura cinzenta e literatura branca?*

3.1.1 Palavras-chave e string de busca

A partir das questões levantadas foi determinado o conjunto mínimo de palavras-chave para a pesquisa. Conjunto esse que foi expandido com base em termos relacionados encontrados durante as etapas de preliminares da busca e antes da aplicação dos filtros.

Esse novo conjunto de termos chave foi então combinado com os operadores OR e AND afim de produzir a string de busca utilizada nas bases selecionadas. O

Palavra-chave	Derivações
rest	restful
best practice	guideline, pattern, anti-pattern
web service	-

Tabela 3.1 Palavras-chave e suas respectivas derivações

uso dos operadores por sua vez aumenta a relevância dos artigos resultantes da pesquisa sem comprometer seu alcance. A string de busca utilizada pode ser vista a seguir como:

("REST"OR "RESTFUL") AND ("BEST PRACTICE"OR "GUIDELINE"OR "PATTERN"OR "ANTI-PATTERN") AND ("WEB SERVICE")

3.1.2 Bases utilizadas

Para coleta da literatura cinzenta, Google foi o motor de busca escolhido e seu espaço de busca foi limitado aos primeiros duzentos resultados retornados. Para a coleta da literatura branca o protocolo de busca foi executado nas seguinte bases:

- ACM (`https://dl.acm.org`)
- Science Direct (`https://www.sciencedirect.com`)
- Emerald (`https://www.emerald.com/insight`)
- Springer (`https://link.springer.com/`)
- IEEE (`https://ieeexplore.ieee.org/`)

3.1.3 Coleta e filtragem

Afim de obter uma seleção de artigos com forte relevância para o tema central foram estabelecidos dois filtros. Sendo o primeiro deles baseado nos critérios de inclusão e exclusão e aplicado a partir dos filtros automatizados disponíveis nas bases de busca e da leitura do título e resumo do artigos selecionados. Enquanto o segundo filtro se baseou em critérios de qualidade pré-definidos e foi avaliado de acordo com a leitura da introdução e conclusão dos artigos selecionados.

3.1.3.1 Critérios de inclusão e exclusão

Os critérios de inclusão e exclusão foram definidos como listado a seguir:

- Critérios de Inclusão
  - Literatura que discute princípios de projetos de APIs REST
  - Literatura que elicita boas práticas em projetos de APIs REST
  - Literatura que elicita padrões em projetos de APIs REST
  - Literatura que elicita anti-padrões em projetos de APIs REST
  - Literatura que explicitamente fazem uso de boas práticas em projetos de APIs REST
- Critérios de Exclusão
  - Artigos não disponíveis em inglês
  - Artigos duplicados
  - Artigos indisponíveis para download ou visualização
  - Artigos que não estejam no período de 2010 a 2020

3.1.3.2 Critérios de qualidade

Durante a filtragem por critérios de qualidade, as introduções e conclusões foram analisadas e escaladas de acordo com os seguintes critérios:

1. Pertinência à área de estudo e tema central
2. Completude e clareza do conteúdo
3. Compatibilidade com as questões de pesquisa

Para cada critério de qualidade foi dada uma nota de 0 a 5 e uma nota geral equivalentes ao somatório da nota obtida três critérios. Os artigos cujo somatório estava acima 7,5 foram selecionados para a revisão. O número de artigos selecionados ao final da filtragem por critérios de qualidade foi 9.

Afim de aumentar o número de artigos relevantes para o mapeamento fora feita a técnica *snowballing*, análise das referências dos artigos já selecionados, sobre os resultados com somatório acima de 10,5. Ao fim do *snowballing*, o número final de artigos foi 15. Na tabela 3.2 é possível acompanhar o processo na literatura branca, seguindo da coleta inicial para filtragem automatizada e leitura dos títulos e resumos. Já na tabela 3.3 é o mesmo processo aplicado às fontes da literatura cinzenta.

Coleta inicial	Primeiro filtro	Segundo filtro	Snowballing
17.775	133	9	15

Tabela 3.2 Resultado quantitativo da coleta nas bases científicas

Coleta inicial	Primeiro filtro	Segundo filtro
200	72	42

Tabela 3.3 Resultado quantitativo da coleta da literatura cinza

3.1.4 Análise da coleta

A partir da leitura do material coletado, foram identificadas 43 práticas direcionadas a APIs REST. Afim de aumentar a relevância das práticas identificadas e separá-las de opiniões individuais sem projeção na comunidade de desenvolvedores foi aplicado mais um filtro. Foram considerados os seguintes critérios de consolidação mutuamente exclusivos:

- Práticas com número de ocorrências na literatura acadêmica maior que 0
- Práticas com número de ocorrências na literatura cinza maior que 4

Após a aplicação dos critérios de consolidação foram selecionadas 25 práticas para compor a pesquisa.

Capítulo 4

Resultados

Neste capítulo serão apresentados os resultados obtidos a partir da coleta e análise das práticas na literatura cinzenta e acadêmica.

4.1 Boas práticas em projetos de APIs REST

Esta seção trata os resultados pertinentes à questão de pesquisa **RQ1**. Para melhor organização dos resultados, as práticas identificadas foram separadas em categorias de acordo com os elementos correspondentes e papel exercido na arquitetura REST.

4.1.1 Boas práticas relacionadas a URI

Nesta subseção são apresentadas práticas de projetos em APIs REST diretamente relacionadas a URI (Universal Resource Identifier).

4.1.1.1 Use substantivos e não verbos na URI

Dada a natureza direcionada a recursos do REST, deve se utilizar substantivos para representação em URIs. Giessler [33] destaca que, além de dificultar a adoção de um sistema de cache, o uso de verbos na URI torna a interface mais próxima de arquiteturas voltadas a método, a exemplo do SOAP. Visto o exemplo na figura 4.1.

```
1 #Ruim
2 /api/users/register
3 /api/register/users
4 /api/createUsers
5
6 #Bom
7 /api/users
```

Figura 4.1 Exemplos de uso incorrento e correto de substantivos na URI.

Código	Prática recomendada	Quantidade de referências	
		Literatura Branca	Literatura Cinzenta
P-01	Use substantivos e não verbos na URI	8	28
P-02	Use a forma plural para descrever recursos na URI	7	26
P-03	Utilize versionamento na API	3	25
P-04	Coleções devem suportar filtragem, ordenação, seleção de campos e paginação	3	23
P-05	Defina sub-recursos de forma hierárquica na URI	3	15
P-06	Evite caracteres especiais e extensões na URI	3	4
P-07	Utilize parâmetros de query string para seleção dos campos de resposta	1	3
P-08	Hífens devem ser usados para melhorar a legibilidade das URIs	1	5
P-09	Letras minúsculas devem ser preferidas em URIs	1	4

Tabela 4.1 Boas práticas relacionadas a URI

4.1.1.2 Use a forma plural para descrever recursos na URI

Para Au-Yeung [9], o uso da forma plural nas URIs reflete de maneira consistente a organização dos recursos em banco de dados. Uma vez que estes também são, de sua própria forma, orientados a coleções e raramente tem apenas uma entrada em suas tabelas. Visto o exemplo na figura 4.2.

```
1 #Ruim
2 /api/user/
3 /api/user/1
4
5 #Bom
6 /api/users/
7 /api/users/1
```

Figura 4.2 Exemplos de uso incorrento e correto da forma plural na URI.

4.1.1.3 Utilize versionamento na API

Utilizada como forma de separar diferentes versões da API e evitar conflitos de compatibilidade [13], o versionamento de APIs REST pode ser realizado de diferentes formas, entre as quais as mais comuns são o uso de uma string indicadora na URI e o uso de cabeçalhos HTTP customizados. Visto o exemplo na figura 4.3.

```
#Versionamento por URI
/api/v1/users
/api/v2/users

#Versionamento por header HTTP
Accept-version: v1
Accept-version: v2
```

Figura 4.3 Exemplos de uso versionamento em APIs REST.

4.1.1.4 Coleções devem suportar filtragem, ordenação e paginação

Por meio de query strings deve-se suportar paginação, ordenação e filtragem em listagens de coleções. Costa reforça que ignorar o uso de paginação pode impactar negativamente a performance da API [16]. Além da economia de banda, o suporte à filtragem e ordenação também reduz a complexidade da implementação do lado do cliente [27]. Visto o exemplo na figura 4.4.

```
#Paginacao
/api/users?page=1
/api/users?page=232

#Ordenacao
/api/users?sort=name
/api/users?sort=age

#Filtragem
/api/users?name=marcos
/api/users?name=jose&username=jose13
```

Figura 4.4 Exemplos de paginação, ordenação e filtragem em URIs.

4.1.1.5 Defina sub-recursos de forma hierárquica na URI

O uso de barras para indicar hierarquia de recursos em URI torna a API mais expressiva e legível. No entanto seu uso traz ressalvas quanto à profundidade da URI em casos de múltiplos níveis de hierarquia. Portanto recomenda-se não ultrapassar o formato: /recurso/identificador/subrecurso. Caso seja necessário consultar uma propriedade de um subrecurso, utilizar uma nova consulta no formato: */recurso/identificador*. Visto o exemplo na figura 4.5.

```
#Ruim
/api/users/19/profiles/1/photo

#Bom
/api/profiles/1/photo
```

Figura 4.5 Exemplos de uso de barras para indicar hierarquia em URIs.

4.1.1.6 Evite caracteres especiais e extensões na URI

De acordo com Alshraideh [8], a presença de símbolos e sublinhados prejudica a legibilidade e compreensibilidade das URIs. No caso de extensões, a necessidade de se customizar o formato de resposta é suprida pela prática de negociação de conteúdo, detalhada mais adiante. Visto o exemplo na figura 4.6.

```
#Ruim
/api/users$/1
/api/articles.json
```

Figura 4.6 Exemplos de uso caracteres especiais e extensões em URIs.

4.1.1.7 Utilize parâmetros de query string para seleção dos campos de resposta

No caso de recursos com representações extensas, uma API que só permite respostas contendo todos os campos do recurso acaba por impactar negativamente a perfomance e funcionalidade da mesma [16]. Para solução de tal cenário recomenda-se o uso de query strings para determinar os campos presentes no retorno. Visto o exemplo na figura 4.7.

4.1.1.8 Hífens devem ser usados para melhorar a legibilidade das URIs

Quando inevitável o uso de termos compostos na URI, indica-se o emprego de hífens para melhoraria da legibilidade [71]. Visto o exemplo na figura 4.8.

```
/api/articles?fields=title,excerpt,date
/api/products?fields=brand,sku,name
```

Figura 4.7 Exemplos do uso de query string para seleção de campos de resposta.

```
#Ruim
/api/best_articles/1
/api/bestArticles/1

#Bom
/api/best-articles/1
```

Figura 4.8 Exemplos de uso incorrento e correto de termos compostos na URI.

4.1.1.9 Letras minúsculas devem ser preferidas em URIs

Uma vez que domínios são tratados de forma case insensitive [61], o emprego de letras minúsculas mantém consistência e promove legibilidade [71]. Visto o exemplo na figura 4.9.

```
#Ruim
/api/PRODUCTS
/api/posts/13/Comments

#Bom
/api/products
/api/posts/13/comments
```

Figura 4.9 Exemplos de uso incorrento e correto letras minúsculas em URIs.

4.1.2 Boas práticas relacionadas a requisição

Nesta subseção são apresentadas práticas de projetos em APIs REST diretamente relacionadas ao processo de requisição e resposta do REST, assim como seu formato de mensagem.

4.1.2.1 Suporte código de status HTTP corretamente

Os códigos de status HTTP, quando empregados corretamente, tornam a comunicação com a API mais semântica e expressiva para o cliente [10]. A tabela 4.3 demonstra o significado mais usual dos códigos de status HTTP em APIs REST.

Código	Prática recomendada	Quantidade de referências	
		Literatura Branca	Literatura Cinzenta
P-10	Suporte códigos de status HTTP corretamente	5	21
P-11	Defina as operações em função dos métodos HTTP	7	18
P-12	Use JSON nas requisições e respostas	3	9
P-13	Use hipertexto para especificar relações	8	7
P-14	O método GET não deve alterar o estado da API	2	6
P-15	Utilize camelCase para nomes de campos no corpo da requisições	0	6
P-16	Use negociação de conteúdo do HTTP	4	3

Tabela 4.2 Boas práticas relacionadas a requisição

4.1.2.2 Defina as operações em função dos métodos HTTP

Segundo Davis [18], afim de fazer uso da natureza semântica dos verbos HTTP e de estalecer um comportamento previsível para o cliente, as operações determinadas pela API REST devem, sempre que possível, seguir o padrão determinado na tabela 4.4.

4.1.2.3 Use JSON nas requisições e respostas

Giessler ressalta[33], que dada sua popularidade e compatibilidade[9], JSON é o formato mais indicado para requisições e respostas. Popularidade atribuída a sua sintaxe menos verborrágico e mais legível. Visto o exemplo na figura 4.10.

```
{"product": {
    "name": "Soap Bar",
    "brand": {
        "name": "Soap Bar Brand"
    },
    "shoppingInfo": {
        "price": "15",
        "amount": "1"
    }
}}
```

Figura 4.10 Exemplo de resposta REST no format JSON.

Código de status HTTP	Caso de uso
200 (OK)	Usado para indicar casos de sucesso não específicos
201 (Created)	Usado para indicar sucesso na criação de um recurso
202 (Created)	Usado para indicar sucesso inicialização de uma operação assíncrona
204 (No Content)	Usado para indicar sucesso quando corpo de resposta for intencionalmente vazio
302 (Found)	Não deve ser usado
304 (Not modified)	Usado para preservar largura de banda
400 (Bad request)	Usado para indicar um erro não específico
401 (Unauthorized)	Usado para indicar erro nas credenciais do cliente
403 (Forbidden)	Usado para proibir acesso independemente do estado das credenciais do cliente
404 (Not found)	Usado quando a URI requisitada pelo cliente não puder ser mapeada para um recurso
405 (Method not allowed)	Usado quando o método HTTP utilizado não for suportado pela API
406 (Not acceptable)	Utilizada quando tipo de mídia (media type) solicitado não puder ser fornecido pela API
409 (Conflict)	Usado para indicar violação no estado do recurso
500 (Internal server error)	Usado para indicar mal funcionamento da API

Tabela 4.3 Boas práticas no uso de código de status HTTP

4.1.2.4 Use HATEOAS para especificar relações

Apesar de fazer parte da conjunto de restrições que define o REST, o HATEOAS ainda não é amplamente utilizado. A restrições trazidas permitem que clientes acessem relações de um recurso sem conhecimento prévio da estrutura da API [16]. Visto o exemplo na figura 4.11.

Método HTTP	Caso de uso
GET	Recupera uma representação de um recurso
HEAD	Recupera os cabeçalhos associados ao recurso
OPTIONS	Recupera a lista de verbos suportados pelo recurso
PUT	Substitui o recurso
DELETE	Apaga o recurso
PATCH	Atualiza o estado do recurso
POST	Usado para criação de um novo recurso e operações genéricas envolvendo envio de dados

Tabela 4.4 Boas práticas no uso de código de status HTTP

```
{"product": {
    "name": "Soap Bar",
    "links": {
        "self": "http://example.com/store/products/1",
        "brand": "http://example.com/store/brands/2"
    },
    "brand": {
        "name": "Soap Bar Brand"
    },
    "shoppingInfo": {
        "price": "15",
        "amount": "1"
    }
}}
```

Figura 4.11 Exemplo de uso do HATEOAS.

4.1.2.5 O método GET não deve alterar o estado da API

A implementação da API deve respeitar o método GET como seguro e idempotente. Sendo um método seguro, aquele livre de efeitos colaterais no recurso acessado e idempotente um método onde ações duplicadas não produzem efeito algum [81].

4.1.2.6 Utilize camelCase para nomes de campos no corpo da requisição

Em conformidade com a estreita relação entre JavaScript e JSON, os campos da requisição devem ser nomeados se utilizando camelCase como padrão de formatação [36]. Visto o exemplo na figura 4.12.

```
#Ruim
{
    "product_name": "Soap Bar",
    "brand-name": "Soap Bar Brand"
}

#Bom
{
    "productName": "Soap Bar",
    "brandName": "Soap Bar Brand"
}
```

Figura 4.12 Exemplo do uso corrento e incorreto de camelCase em mensagens REST.

4.1.2.7 Use negociação de conteúdo HTTP

Negociação de conteúdo é um mecanismo capaz de, para uma mesma URI, servir diferentes formatos de resposta. O formato do conteúdo é negociado, principalmente, com o uso do cabeçalho HTTP Accept. O exemplo 4.13 traz diferentes possibilidades de preenchimento do cabeçalho.

```
Accept: text/html
Accept: application/json
Accept: image/png
Accept: audio/wav
```

Figura 4.13 Exemplo de uso do cabeçalho *Accept*.

4.1.3 Boas práticas relacionadas a tratamento de erros

Nesta subseção são apresentadas práticas de projetos em APIs REST relacionadas a tratamentos e recuperação de erros.

Código	Prática recomendada	Quantidade de referências	
		Literatura Branca	Literatura Cinzenta
P-17	Especifique erros com os código de status HTTP	5	16
P-18	Use mensagens de erro expressivas	2	16
P-19	Trate casos de barra no final da URL com redirecionamento	1	1

Tabela 4.5 Boas práticas relacionadas a tratamento de erros

4.1.3.1 Especifique erros com códigos de status HTTP

Segundo Alshraiedeh [8], é vista uma tendência dos desenvolvedores em ignorar os códigos de status diferentes de 2xx. Assim criando um cenário mais sucestível a erros e exigindo maior cuidado no tratamento dos possíveis retorno da API.

4.1.3.2 Use mensagens de erro expressivas

Em conjunto com o código de status HTTP, a mensagem de erro faz parte das ferramentas à disposição do cliente quando algum comportamento inesperado ocorre na API. Portanto é essencial compor o retorno de uma situação de erro com um corpo em formato JSON pré-definido e uma mensagem descrevendo precisamente a causa da exceção. Visto o exemplo na figura 4.14.

```
#Ruim
{
    "error": "A error occurred",
    "message": "A error occurred"
}

#Bom
{
    "error": "Not Found",
    "message": "The product  with ID 135 couldn't be found"
}
```

Figura 4.14 Exemplos incorrentos e corretos de mensagens de erro.

4.1.3.3 Trate casos de barra no final da URL com redirecionamento

Um erro comum em acesso à URLs disponíveis em API é a inconsistência em relação ao uso de barra ao final da URI. Duas URIs com o caminho idêntico são tratadas pelo HTTP como dois recursos diferentes, ocasionando erros de recurso não encontrado. Para evitar tal problema, é indicado decidir entre a presença ou ausência da barra e tratar os casos incompatível com redirecionamento para a URI correta [71].

4.1.4 Outras boas práticas

Nesta subseção são apresentadas o restante das práticas de projetos em APIs REST que não se relacionam com as categorias anteriores.

Código	**Prática recomendada**	**Quantidade de referências**	
		Literatura Branca	**Literatura Cinzenta**
P-20	Use SSL/TLS	1	10
P-21	Documentação deve estar publicamente acessível	2	10
P-22	Limite as requisições às APIs	1	5
P-23	Utilize cache para requisições	4	7
P-24	Documentação deve seguir a especificação da OpenAPI	0	7
P-25	Use OAuth	3	7

Tabela 4.6 Outras boas práticas

4.1.4.1 Use SSL/TLS

Como forma de proteger as informações trocadas entre API e cliente de terceiros, o SSL/TLS é empregado. Parte essencial da segurança de grandes APIs comerciais [13], o SSL/TLS pode ser facilmente implementado dado o extenso suporte entre os soluções de serviço HTTP.

4.1.4.2 Documentação deve estar publicamente acessível

Facilidade de descoberta com o uso de palavras-chave e cobertura de todos os casos de uso e mudanças de estado de recurso são caraterísticas base de uma boa documentação.

4.1.4.3 Limite as requisições à API

Como forma de evitar uso abusivo de banda proveniente da ação de clientes maliciosos ou até pico de uso legítimo de determinados serviços, é recomendado o emprego de limite à requisições por meio do cabeçalho X-RateLimit-Limit. Para casos de limite excedido é recomendado o uso do código de status 429 (Too many requests).

4.1.4.4 Utilize cache para requisições

Unindo melhorias em performance à economia de requisições ao banco de dados, o uso de cache traz um menor tempo de resposta aos clientes da API. Vale ressaltar que as restrições definidas por Fielding [24] na especificação original do REST já traziam cache como elemento chave.

4.1.4.5 Documentação deve seguir a especificação da OpenAPI

DuVander [21] reforça que contando com a estrutura e flexibilidade de ferramentas que suportam a especificação do OpenAPI, APIs são capazes de rapidamente gerar documentações completas e interativas (Swagger) ou testar serviços por meios de mock servers (Prism).

4.1.4.6 Use OAuth

Inicialmente criado para delegação de acesso à APIS, OAuth é um padrão aberto para autorização capaz de proteger recursos de acessos não autorizados. Dentre os métodos de identificação e autenticação, Bülthoff [13] ressalta a expressiva adoção do OAuth.

4.2 Atributos de qualidade em projetos de APIs REST

Esta seção trata os resultados pertinentes à questão de pesquisa **RQ2**.

4.2.1 Atributos de qualidade de serviços web

Esta subseção apresenta os atribuitos de qualidade de software aplicados a serviços web e utilizados para categorização das práticas identificadas.

Interoperabilidade (Interoperability)
: Interoperabilidade descreve a capacidade de um sistema de estabelecer comunicação com um ou mais serviços web em um ambiente heterogêneo [48].

Confiabilidade (Reliability)
Confiabilidade descreve a capacidade de um sistema de permanecer livre de erros [60].

Segurança (Security)
Segurança descreve a capacidade de um sistema impedir acessos indevidos a recursos privados.

Testabilidade (Testability)
Testabilidade representa o grau um sistema suporta o próprio teste sem necessidade de maiores alterações [30].

Performance (Performance)
Performance descreve a capacidade de um serviço de responder corretamente uma requisição em curto espaço de tempo de maneira eficiente [16].

Detectabilidade (Discoverability)
Detectabilidade descreve a capacidade de um sistema de ser descoberto por outra aplicação ou serviço [45].

Disponibilidade (Availability)
Disponibilidade descreve a capacidade de um sistema de permanecer funcionando conformo o programado pela maior tempo possível [4].

Modificabilidade (Modifiability)
Modificabilidade descreve a capacidade de um sistema de ser alterado com baixo custo e risco [20].

Proteção (Safety)
A capacidade de proteção de um serviço web se traduz na seu suporte à idempotência dos métodos HTTP. Isto é, ausência de efeitos colaterais para os métodos considerados seguros [16].

Funcionalidade (Functionality)
Funcionalidade descreve a oferta de funções afim de customizar o comportamento de um determinada sistema para um determindado caso de uso. [16].

Implantabilidade (Deployability)
Implantabilidade descreve a facilidade de implantação de um sistema em determinado ambiente de execução [16].

4.2.2 Mapeamento de práticas

Esta subseção apresenta o resultado do mapeamento de práticas em projetos de APIs REST sobre os atributos de qualidade de software levantados.

A tabela 4.7 revisita as práticas elencadas na seção anterior e as categoriza de acordo com o atributo de qualidade majoritariamente afetado pela aplicação da mesma.

Atributo de qualidade	Práticas
Interoperabilidade (Interoperability)	P-12, P-13, P-15, P-16
Confiabilidade (Reliability)	P-19
Segurança (Security)	P-17, P-20, P-25
Testabilidade (Testability)	P-18, P-24
Performance (Performance)	P-22, P-23
Detectabilidade (Discoverability)	P-01, P-02, P-05, P-06, P-08, P-09, P-21
Disponibilidade (Availability)	-
Modificabilidade (Modifiability)	P-03
Proteção (Safety)	P-14
Funcionalidade (Functionality)	P-04, P-07, P-10, P-11
Implantabilidade (Deployability)	-

Tabela 4.7 Distribuição das práticas de acordo com atributos de qualidade de software

Para melhor visualização das proporções a figura 4.15 traz os dados da tabela 4.7 de forma agregada. Destaca-se, na figura, o alto volume de referências em interoperabilidade, detectabilidade e funcionalidade.

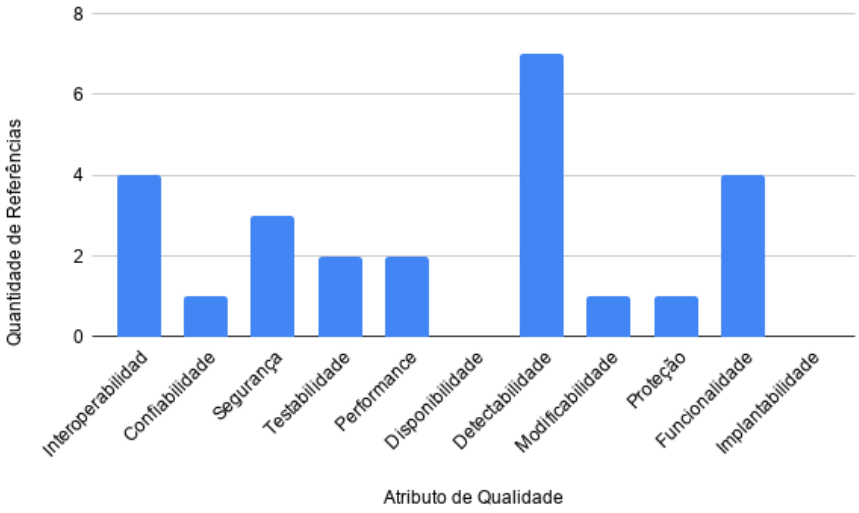


Figura 4.15 Distribuição das práticas recomendadas por atributos de qualidade.

As figuras 4.16 e 4.17 trazem as distribuições das referências coletadas na literatura branca e cinzenta pelos atributos de qualidade avaliados. Quando compara-

das, as distribuições revelam um maior referências discutindo práticas de interoperabilidade na literatura branca quando comparada à literatura cinzenta. Isso se deve, principalmente, ao grande número de referências acadêmicas promovendo o uso de HATEOAS como prática recomendada.

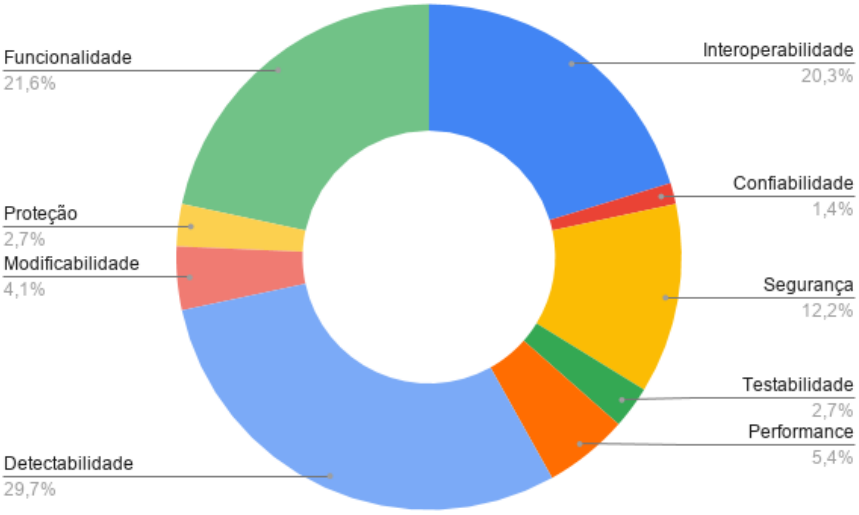


Figura 4.16 Gráfico de distribuição das referências da literatura branca de acordo com atributos de qualidade.

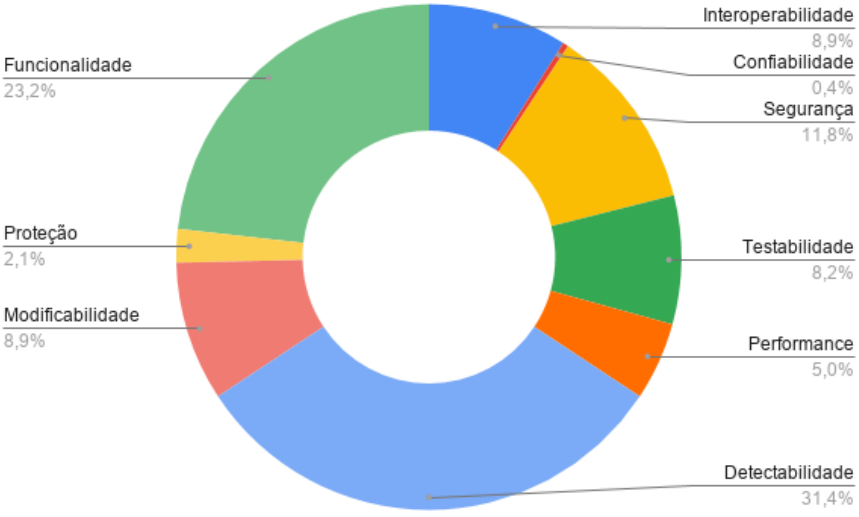


Figura 4.17 Gráfico de distribuição das referências da literatura cinzenta de acordo com atributos de qualidade.

4.3 Distribuição e tendências na literatura branca e cinzenta

Esta seção trata os resultados pertinentes à questão de pesquisa **RQ3**

4.3.1 Análise por categoria

Revisitando as categorias utilizadas na primeira seção deste capítulo, as figuras 4.18 e 4.19 ilustram a distribuição das referências na literatura branca e cinzanta de acordo com a funcionalidade ou elemento REST relacionado. Os gráficos, por sua vez, demonstram um padrão equivalente de distribuição entre os dois tipos de literatura.

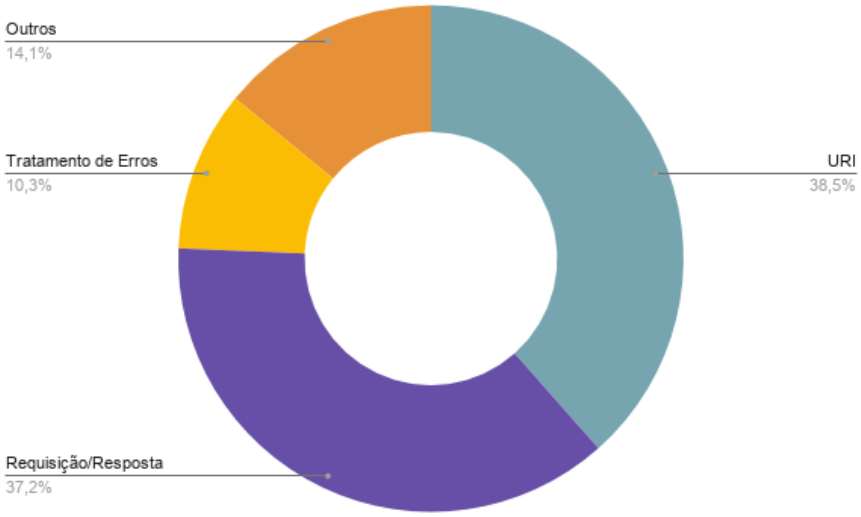


Figura 4.18 Distribuição de referências na literatura branca por funcionalidade.

4.3.2 Análise por ano de publicação

Afim de identificar tendências de crescimento e decrescimento, as figuras 4.20 e 4.21 trazem série temporais demonstrando o avanço da produção sobre boas práticas em projetos de APIs REST nos últimos dez anos. Fica evidente o alto volume de material recentemente produzido na literatura cinzenta, enquanto a literatura branca demonstra um crescimento maior entre os anos de 2014 e 2016.

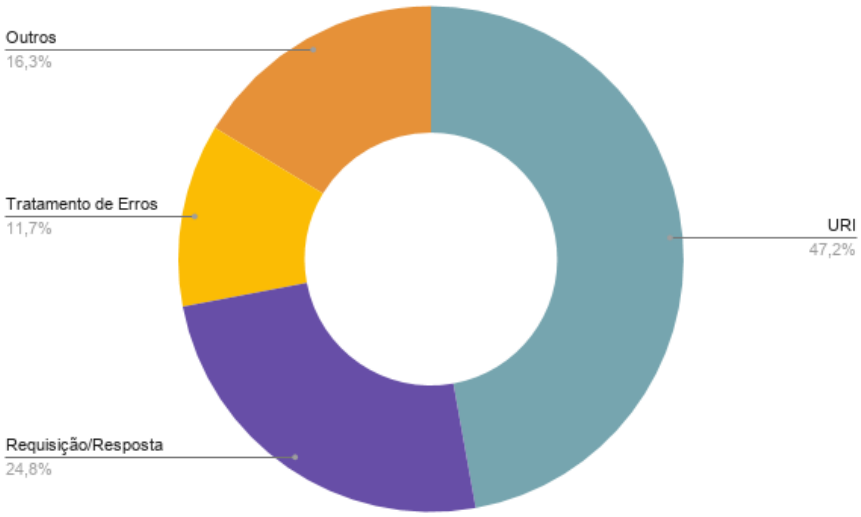


Figura 4.19 Distribuição de referências na literatura cinzenta por funcionalidade.

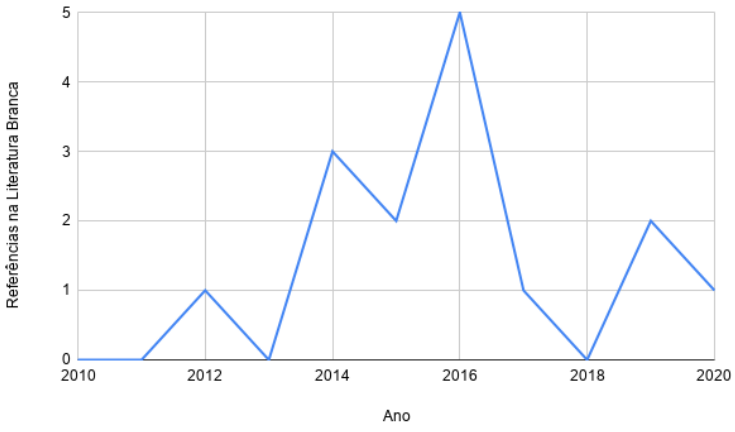


Figura 4.20 Quantidade de referências na literatura branca entre 2010 e 2020.

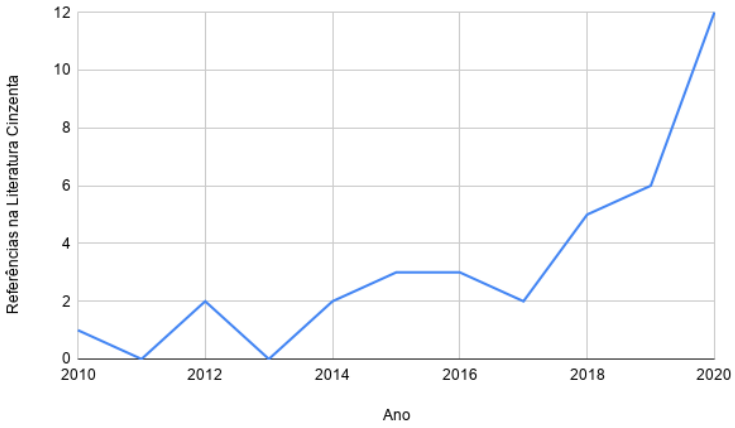


Figura 4.21 Quantidade de referências na literatura cinzenta entre 2010 e 2020.

Capítulo 5

Considerações Finais

Este capítulo apresenta as considerações finais do trabalho. Serão apresentadas as recomendações para trabalhos futuros e as conclusões obtidas com a pesquisa.

5.1 Trabalhos futuros

Serviços *web* e sua APIs são ainda peças fundamentais das aplicações com as quais interagimos diariamente. Essa tendência não demonstra sinais de enfraquecimento, mas traz novos métodos como alternativas às tecnologias já estabelecidas. Este é o caso do GraphQL, que recentemente vem se mostrando uma alternativa viável ao REST em aplicações de *web* dado o fato da primeira, diferente do REST, ser uma linguagem de consulta não restrita a uma representação única de recurso e já incluir em seu funcionamento intrínseco práticas levantadas nesse trabalho. É sugerido como trabalho futuro, um mapeamento de práticas GraphQL em fontes da literatura branca e cinzenta.

É sugerida também a possibilidade de expansão do trabalho atual por meio do desenvolvimento de uma ferramenta de detecção automatizada de anti-padrões REST utilizando como base a metodologia de coleta executada nesta pesquisa.

5.2 Conclusão

Este trabalho teve como objetivo observar as diferenças nas práticas recomendadas para projetos de APIs REST encontradas na literatura branca e cinzenta.

A primeira diferença, evidenciada na coleta, se deu na dificuldade de encontrar artigos relevantes na literatura branca. Tal evidência vai de encontro ao já conhecido distanciamento entre a produção prática e o conhecimento téorico. A segunda diferença observada se deu na ausência de suporte ao padrão *OpenAPI* por parte das referências na literatura branca. A ausência é sentida de forma negativa uma vez que o OpenAPI tem se consolidade como referência na indústria quando o assunto é especificação de APIs e serviços web. Seu uso em um projeto de API REST é forte indicativo de compatibilidade com ferramentas de teste e nível de detalhamento da documentação.

A terceira diferença observada se deu na maior concetração de referências a

práticas que afetam interoperabilidade na literatura branca. Sendo essa concentração diretamente relacionada a forte presença de referências à restrição HATEOAS na literatura branca. A razão específica desta forte ocorrência, no entanto, vai além do escopo deste trabalho. A quarta diferença observada se deu na inconstância da produção sobre práticas REST na literatura branca em total oposto ao crescente volume de artigos provenientes da literatura cinzenta no decorrer dos últimos dez anos.

Em uma última análise é visto que apesar do menor volume de produções na literatura branca, pôde se observar uma cobertura de práticas muito próxima do levantado na literatura cinzenta. Isto é, das 25 práticas levantadas 23 estavam presentes nas referências brancas, totalizando 92% das práticas recomendadas em projetos de APIs REST.

Referências Bibliográficas

[1] Rfc for rest? https://web.archive.org/web/20091111012314/http://tech.groups.yahoo.com/group/rest-discuss/message/6757, nov 2006. (Acesso em: 2020-10-12).

[2] Use restful service urls. https://apiguide.readthedocs.io/en/latest/build_and_publish/use_RESTful_urls.html, sep 2015. (Acesso em: 2020-10-12).

[3] Restful parameters antipattern considerations for queries, paths. https://www.theserverside.com/opinion/RESTful-parameters-antipattern-considerations-for-queries-paths, feb 2019. (Acesso em: 2020-10-12).

[4] Abraham, S., Thomas, M., and Thomas, J. Enhancing web services availability. In *IEEE International Conference on e-Business Engineering (ICEBE'05)* (2005), IEEE.

[5] Albano, J. Best practices for rest api error handling. https://www.baeldung.com/rest-api-error-handling-best-practices, jun 2020. (Acesso em: 2020-10-12).

[6] Alshraiedeh, F. S., and Katuk, N. A URI parsing technique and algorithm for anti-pattern detection in RESTful web services. *International Journal of Web Information Systems ahead-of-print*, ahead-of-print (Nov. 2020).

[7] AMBRA, J. Web api design: 5 best practices to know. https://www.toptal.com/api-developers/5-golden-rules-for-designing-a-great-web-api. (Acesso em: 2020-10-12).

[8] and, F. A. SOAP and RESTful web service anti-patterns: A scoping review. *International Journal of Advanced Trends in Computer Science and Engineering* (Oct. 2019), 1831–1840.

[9] Au-Yeung, J. Best practices for rest api design - stack overflow blog. https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/, mar 2020. (Acesso em: 2020-10-12).

[10] Brabra, H., Mtibaa, A., Petrillo, F., Merle, P., Sliman, L., Moha, N., Gaaloul, W., Guéhéneuc, Y.-G., Benatallah, B., and Gargouri, F. On semantic detection of cloud API (anti)patterns. *Information and Software Technology 107* (Mar. 2019), 65–82.

[11] Brabra, H., Mtibaa, A., Sliman, L., Gaaloul, W., Benatallah, B., and Gargouri, F. Detecting cloud (anti)patterns: OCCI perspective. In *Service-Oriented Computing*. Springer International Publishing, 2016, pp. 202–218.

[12] Brylinski, A.-S., and Bhattacharjya, A. Overview of http/2. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing* (New York, NY, USA, 2017), ICC '17, Association for Computing Machinery.

[13] Bülthoff, F., and Maleshkova, M. RESTful or RESTless – current state of today's top web APIs. In *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 64–74.

[14] Bush, T. 10+ best practices for naming api endpoints. `https://nordicapis.com/10-best-practices-for-naming-api-endpoints/`, jan 2020. (Acesso em: 2020-10-12).

[15] Constine, J. Facebook now has 2 billion monthly users... and responsibility | techcrunch. `https://techcrunch.com/2017/06/27/facebook-2-billion-users/`, jun 2017. (Acesso em: 2020-10-12).

[16] Costa, B., Pires, P. F., Delicato, F. C., and Merson, P. Evaluating REST architectures—approach, tooling and guidelines. *Journal of Systems and Software 112* (Feb. 2016), 156–180.

[17] Cycles, A. https://www.apiopscycles.com/rest-api-design-guide. `https://www.apiopscycles.com/rest-api-design-guide`. (Acesso em: 2020-10-12).

[18] Davis, C. What if the web were not RESTful? In *Proceedings of the Third International Workshop on RESTful Design - WS-REST '12* (2012), ACM Press.

[19] Delamore, S. Web api best practices | quick answers for designing rest services. `https://www.genui.com/resources/rest-web-api-best-practices`, jun 2020. (Acesso em: 2020-10-12).

[20] Desruelle, H., and Gielen, F. Architectural modifiability considerations for designing a multi-device web application platform. *Procedia Computer Science 19* (2013), 895–900.

[21] DuVander, A. Api design patterns | restful api web services design | rest api design. https://stoplight.io/blog/api-design-patterns-for-rest-web-services/, feb 2020. (Acesso em: 2020-10-12).

[22] Ferlin, A. Restful api patterns. https://levelup.gitconnected.com/restful-api-patterns-81930c43e494, jan 2020. (Acesso em: 2020-10-12).

[23] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol – HTTP/1.1. Tech. rep., June 1999.

[24] Fielding, R. T., and Taylor, R. N. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, 2000. AAI9980887.

[25] Foundation, M. A typical http session - http. https://developer.mozilla.org/en-US/docs/Web/HTTP/Session. (Acesso em: 2020-10-02).

[26] Foundation, W. W. W. History of the web – world wide web foundation. https://webfoundation.org/about/vision/history-of-the-web/, 2020. (Acesso em: 2020-10-12).

[27] Fredrich, T. Restful service best practices. Tech. rep., Pearson eCollege, oct 2012. (Acesso em: 2020-10-12).

[28] Fredrich, T. Rest api tutorial. https://www.restapitutorial.com/, 2019. (Acesso em: 2020-10-12).

[29] Fried, I. Google's g suite cracks 2 billion users. https://www.axios.com/google-g-suite-total-users-9a6d3df6-c990-4866-9efc-ba6756ba3c4d.html, mar 2020. (Acesso em: 2020-10-12).

[30] Garousi, V., Felderer, M., and Kılıçaslan, F. N. A survey on software testability. *Information and Software Technology 108* (Apr. 2019), 35–64.

[31] Garousi, V., Felderer, M., and Mäntylä, M. V. The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (New York, NY, USA, 2016), EASE '16, Association for Computing Machinery.

[32] Garousi, V., Felderer, M., and Mäntylä, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology 106* (Feb. 2019), 101–121.

[33] Giessler, P., Gebhart, M., Sarancin, D., Steinegger, R., and Abeck, S. Best practices for the design of restful web services. *Proceedings - International Conference on Software Engineering 10* (11 2015), 392–.

[34] Gottschalk, K., Graham, S., Kreger, H., and Snell, J. Introduction to web services architecture. *IBM systems Journal 41*, 2 (2002), 170–177.

[35] Gourley, D. *HTTP : the definitive guide*. O'Reilly, Beijing Sebastopol, CA, 2002.

[36] Haldar, M. Restful api designing guidelines — the best practices. https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9, mar 2016. (Acesso em: 2020-10-12).

[37] Halili, F., and Ramadani, E. Web services: A comparison of soap and rest services. *Modern Applied Science 12*, 3 (Feb. 2018), 175.

[38] Harrison, P. The essential guide for designing a production-ready, developer-friendly restful api. https://blog.logrocket.com/the-essential-guide-for-designing-a-production-ready-developer-friendly-rest oct 2019. (Acesso em: 2020-10-12).

[39] Hauer, P. Restful api design. best practices in a nutshell. https://phauer.com/2015/restful-api-design-best-practices/, mar 2015. (Acesso em: 2020-10-12).

[40] Hegoda, D. Rest api architecture - best practices. https://dasunhegoda.com/rest-api-architecture-best-practices/1049/, nov 2015. (Acesso em: 2020-10-12).

[41] House, C. How restful is your api? https://www.bitnative.com/2012/08/26/how-restful-is-your-api/, aug 2012. (Acesso em: 2020-10-02).

[42] IBM. Rest api guidelines created for the watson developer cloud services. https://github.com/watson-developer-cloud/api-guidelines, 2020. (Acesso em: 2020-10-12).

[43] IBM. What is a web service? https://www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html, jul 2020. (Acesso em: 2020-10-12).

[44] Imperva. Web api security | best practices for soap and rest api | imperva. https://www.imperva.com/learn/application-security/web-api-security/. (Acesso em: 2020-10-12).

[45] Isikdag, U. Foundational SOA patterns for complex information models. In *Enhanced Building Information Models*. Springer International Publishing, 2015, pp. 25–41.

[46] Jain, S. Guidelines & best practices for design restful api. https://dev.to/sachinjain007/guidelines-best-practices-for-design-restful-api-5575, oct 2019. (Acesso em: 2020-10-12).

[47] JAUKER, S. 10 best practices for better restful api. https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291, jun 2014. (Acesso em: 2020-10-12).

[48] Johari, K., and Kaur, A. Interoperability issues in web services. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology - CCSEIT '12* (2012), ACM Press.

[49] Kalin, M. *Java Web services : up and running*. O'Reilly, Beijing, 2013.

[50] Kapadnis, J. Rest: Good practices for api design. https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9, mar 2018. (Acesso em: 2020-10-12).

[51] Karanam, R. Rest api best practices — design examples from java and spring web services. https://dzone.com/articles/rest-api-best-practices-with-design-examples-from, nov 2019. (Acesso em: 2020-10-12).

[52] Kumar, D. Best practices for building restful web services. Tech. rep., Infosys Limited, 2018.

[53] Lange, K. Don't limit your rest api to crud operations. https://www.kennethlange.com/dont-limit-your-rest-api-to-crud-operations/, nov 2018. (Acesso em: 2020-10-12).

[54] Levin, G. 5 basic rest api design guidelines. https://blog.restcase.com/5-basic-rest-api-design-guidelines/, oct 2016. (Acesso em: 2020-10-12).

[55] Manca, F. Restful api design: 13 best practices to make your users happy. https://florimond.dev/blog/articles/2018/08/restful-api-design-13-best-practices-to-make-your-users-happy/, aug 2018. (Acesso em: 2020-10-12).

[56] Microsoft. Api design guidance - best practices for cloud applications. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design, jan 2018. (Acesso em: 2020-10-12).

[57] Microsoft. Microsoft rest api guidelines. https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md, 2020. (Acesso em: 2020-10-12).

[58] Mishra, D. Top 6 rest naming best practices. https://metamug.com/article/rest-api-naming-best-practices.html, jun 2020. (Accessed on 11/21/2020).

[59] Montes, J. C. Best practices for your rest api. https://solidgeargroup.com/en/best-practices-rest-api/, oct 2017. (Acesso em: 2020-10-12).

[60] Moser, L., Melliar-Smith, P., and Zhao, W. Building dependable and secure web services. *Journal of Software 2* (02 2007).

[61] Motorola. Domain Name System (DNS) Case Insensitivity Clarification. RFC 4343, RFC Editor, jan 2006.

[62] Mulders, M. 13 best practices for building restful apis. https://www.sitepoint.com/build-restful-apis-best-practices/, aug 2020. (Acesso em: 2020-10-12).

[63] Mulloy, B. Web api design. Tech. rep., Apigee, mar 2012.

[64] Muschamp, P. An introduction to web services. *BT Technology Journal 22*, 1 (2004), 9–18.

[65] OWASP. Rest security - owasp cheat sheet series. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html, jul 2020. (Acesso em: 2020-10-12).

[66] Palma, F., Dubois, J., Moha, N., and Guéhéneuc, Y.-G. Detection of REST patterns and antipatterns: A heuristics-based approach. In *Service-Oriented Computing*. Springer Berlin Heidelberg, 2014, pp. 230–244.

[67] Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.-G., and Tremblay, G. Are RESTful APIs well-designed? detection of their linguistic (anti)patterns. In *Service-Oriented Computing*. Springer Berlin Heidelberg, 2015, pp. 171–187.

[68] Papazoglou, M. *Web services : principles and technology*. Pearson/Prentice Hall, Harlow, England New York, 2008.

[69] Pautasso, C., Ivanchikj, A., and Schreier, S. A pattern language for RESTful conversations. In *Proceedings of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16* (2016), ACM Press.

[70] Pecanac, V. Top rest api best practices. `https://dzone.com/articles/top-rest-api-best-practices`, sep 2020. (Acesso em: 2020-10-12).

[71] Petrillo, F., Merle, P., Moha, N., and Guéhéneuc, Y.-G. Are REST APIs for cloud computing well-designed? an exploratory study. In *Service-Oriented Computing*. Springer International Publishing, 2016, pp. 157–170.

[72] Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., and Percannella, G. REST APIs: A large-scale analysis of compliance with principles and best practices. In *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 21–39.

[73] Sahni, V. Best practices for designing a pragmatic restful api. `https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api`. (Acesso em: 2020-10-12).

[74] Sidorenko, V. Restful api design: Best practices - gearheart. `https://gearheart.io/blog/restful-api-design-best-practices/`, oct 2017. (Acesso em: 2020-10-12).

[75] Stempniak, M. 9 best practices to implement in rest api development. `https://www.merixstudio.com/blog/best-practices-rest-api-development/`, jun 2019. (Acesso em: 2020-10-12).

[76] Subramaniam, P. Rest api design - resource modeling. `https://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling`, aug 2014. (Acesso em: 2020-10-12).

[77] Tabbaa, B. Anti-patterns of web api's. `https://itnext.io/anti-patterns-of-web-apis-33d2a28534f2`, jan 2018. (Acesso em: 2020-10-12).

[78] Vasudevan, K. Best practices in api design. `https://swagger.io/blog/api-design/api-design-best-practices/`, oct 2016. (Acesso em: 2020-10-12).

[79] Vitvar, T. Api anti-patterns: How to avoid common rest mistakes | programmableweb. `https://www.programmableweb.com/news/api-anti-patterns-how-to-avoid-common-rest-mistakes/2010/08/13`, aug 2010. (Acesso em: 2020-10-12).

[80] Wagh, D. K., and Thool, R. A comparative study of soap vs rest web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications 2* (07 2012), 12–16.

[81] Yuan, T., Tang, Y., Wu, X., Zhang, Y., Zhu, H., Guo, J., and Qin, W. Formalization and verification of REST on HTTP using CSP. *Electronic Notes in Theoretical Computer Science 309* (Dec. 2014), 75–93.

[82] Zalando. Zalando restful api and event scheme guidelines. `https://opensource.zalando.com/restful-api-guidelines/`, oct 2020. (Acesso em: 2020-10-12).