APIs RESTful

Orientações Gerais

Histórico da Revisão

Data	Versão	Descrição	Autor
22/03/2017	0.1	Criação do documento	ARQDIGITAL
03/04/2017	0.2	Revisão	ARQDIGITAL
06/04/2017	0.3	Revisão	ARQDIGITAL
18/04/2017	1.0	Versão Final	ARQDIGITAL
28/12/2017	1.1	Revisão, inclusão e detalhamento dos padrões	César Lima Piau

Índice Analítico

1 INTRODUÇÃO

1.1 APRESENTAÇÃO E OBJETIVO DO PADRÃO

Este documento fornece orientações gerais para desenvolvimento de Web APIs da Caixa.

O objetivo deste documento é fomentar a qualidade, consistência e manutenibilidade das APIs, bem como agilizar o desenvolvimento e pacificar discussões quanto ao uso de padrões e melhores práticas.

O desenvolvimento de APIs na CAIXA tem como objetivo fornecer uma forma simplificada de publicação e consumo de web services, alcançando assim uma grande variedade de clientes em diversas tecnologias.

1.2 PÚBLICO ALVO

Desenvolvedores e Consumidores de APIs.

2 PADRÕES DE DESENVOLVIMENTO

Uma Web API é um tipo de serviço (web service) que é exposto utilizando basicamente o protocolo HTTP.

Portanto, o protocolo HTTP é o alicerce das Web APIs, o que significa que durante o design de uma API, devemos lidar com todos os aspectos do protocolo (URL, Métodos, Cabeçalhos e Status Code).

O padrão arquitetural que utiliza o protocolo HTTP para o desenvolvimento de Web APIs é chamado de REST (Representational State Transfer) e as APIs desenvolvidas utilizando esse padrão arquitetural são chamadas de APIs RESTful.

A seguir serão descritos os principais padrões e melhores práticas que envolvem o desenvolvimento de APIs RESTful.

2.1 PADRÃO PARA URLs

A definição da URL base é uma das partes mais importantes do design de uma API, pois boa parte do entendimento de como a API funciona pode ser obtido interpretando a sua URL.

É importante que a URL tenha um domínio de negócio que permita agrupar as APIs de forma coerente. Exemplo:

```
https://api.caixa/cartao-credito/v1/faturas
https://api.caixa/cadastro/v1/clientes
```

As URLs devem representar recursos no formato “lista/item” e devemos utilizar substantivos (não verbos) para identificar tais recursos. O nome das listas devem estar sempre no plural. Exemplo:

```
https://api.caixa/cartao-credito/v1/faturas/id-fatura
https://api.caixa/cadastro/v1/clientes/id-cliente
```

Recomenda-se aninhar objetos em no máximo 2 níveis. Exemplo:

```
https://api.caixa/cadastro/v1/clientes/id-cliente/enderecos
```

Para nomes compostos de recursos ou atributos na URL, deve-se utilizar o padrão **spinal-case** ou, caso exista algum impedimento, utilizar o padrão **cramcase**. A sintaxe das URL deve seguir a **RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax**.

As APIs devem ser publicadas com um número de versão, pois o versionamento proporciona iterações mais rápidas, evita pedidos inválidos e mantém endpoints atualizados, além de permitir que as aplicações mais antigas consumam as primeiras versões da API e continuem funcionando corretamente.

As versões devem utilizar números inteiros prefixados com 'v', como por exemplo: v1, v2, v3. O uso de versões fracionadas deve ser evitado e a versão só deve ser incrementada se houver mudança na interface. A versão da API deve ser independente da versão do provedor do serviço, seja ele uma aplicação Java ou um fluxo no Barramento de Serviços.

O padrão recomendado para a URL de uma API seria o seguinte:

```
https://api.caixa/{domínio-de-negócio}/{versão}/{lista-plural}/{item}
```

2.2 USO DE QUERY STRINGS

As query strings são componentes importantes das APIs, permitindo a inserção de parâmetros que podem ser utilizados em diferentes situações.

2.2.1 FILTROS

Em APIs de consulta, os filtros possibilitam que apenas os registros necessários para a aplicação consumidora sejam trafegados.

É possível filtrar uma lista por vários atributos ao mesmo tempo e filtrar mais de um valor para um atributo. Exemplo:

```
http://api.caixa/informacoes/v1/unidades?uf=DF,MG&penhor=sim
```

2.2.2 CONTEÚDO PARCIAL

Nos casos em que um recurso possui muitos atributos, a API pode permitir a consulta de conteúdo parcial. Exemplo:

```
https://api.caixa/cadastro/v1/clientes/id_cliente?campos=nome,idade
```

2.2.3 PAGINAÇÃO

A paginação de resultados é uma opção quando o volume de registros de uma consulta for muito grande.

Para a paginação, deve-se utilizar os parâmetros **offset** e **limit**, onde offset representa o índice do primeiro elemento da lista e limit a quantidade de elementos a serem retornados na lista.

A resposta da API para uma lista paginada poderá utilizar o status HTTP **206 – Partial Content**. É importante que a resposta também contenha referências para a página anterior e a próxima página. Exemplo:

```
http://api.caixa/informacoes/v1/unidades?limit=3

HTTP 206 OK
{
“unidades”:[
  {"Codigo":"0002", "nome":"PLANALTO, DF"},
  {"Codigo":"0003", "nome":"AEROPORTO PRESIDENTE JK, DF"},
  {"Codigo":"0004", "nome":"BERNARDO SAYAO, DF"}],
“paginacao”:{
  “count”: 100,
  “next”:”http://api.caixa/v1/unidades?offset=3,limit=3”}
}
```

2.2.4 ORDENAÇÃO

Deve-se evitar o uso de ordenação no servidor e dar preferência, sempre que possível, para que a ordenação dos resultados seja feita no cliente.

Caso seja necessária a implementação de ordenação de resultados no servidor, deve-se usar os parâmetros **sort** e **desc**, onde sort contém o nome do atributo usado para ordenar a lista e desc indica que a ordem será descendente ou decrescente (por padrão a ordem é ascendente ou crescente). Exemplo:

```
https://api.caixa/cadastro/v1/clientes?sort=nome
```

2.3 USO CORRETO DOS METODOS HTTP

Uma vez que as APIs na maioria dos casos representam recursos em vez de ações, devemos utilizar os verbos ou métodos HTTP para operar esses recursos.

Dentro do universo de métodos HTTP, os mais utilizados para as APIs são os seguintes:

MÉTODO HTTP	DESCRIÇÃO
GET	Método utilizado para consultar registros podendo encaminhar na URL da requisição filtros para recuperação dos dados.
POST	Método utilizado para criar registros. Os dados do registro a ser criado podem ser passados na URL como parâmetros ou no corpo da requisição em formatos como JSON, XML e outros.
PUT	Método usado para atualizar registros ou para criá-los quando o ID do recurso é escolhido pelo cliente, e não pelo servidor. O corpo da requisição

	contém uma representação do registro podendo estar em formatos como XML, JSON e outros.
DELETE	Método utilizado para remover um registro. Os dados do registro a ser criado são passados na URL como parâmetros.
OPTIONS	Método utilizado para consultar ao servidor quais os métodos, cabeçalhos e origens aceitas para uma determinada API. Muito utilizado em requisições cross-origin (CORS).

Apesar do padrão REST preconizar o uso dos métodos HTTP para representar as operações, existem casos em que será necessário adequar o modelo para atender alguma necessidade específica.

Um exemplo muito comum que foge à regra é uma API de login que, além de conter um verbo na sua URL, ainda pode utilizar o método POST para passar os dados sensíveis de autenticação dentro do corpo da requisição. Exemplo:

```
POST
https://api.caixa/login-caixa/v1/login
{“usuário”:”x”,”senha”:”y”}
```

Outra necessidade de utilização de métodos HTTP diferente do indicado é nos casos em que uma consulta precisa ser feita com um identificador ou filtro que contenha informações sensíveis. Nesse caso, deve-se utilizar o método POST, indicar na URL a ação que pretende fazer e passar os dados sensíveis no corpo da requisição. Exemplo:

```
POST
https://api.caixa/cartao-credito/v1/faturas/consulta
{“numero_cartao”:”9999.9999.9999.9999”, “mes”:”01”, “ano”:”2017”}
```

2.4 TIPO DE RETORNO DE DADOS

O formato JSON é o formato padrão para a troca de mensagens em uma API, mas é importante que a API seja capaz de responder em mais de um formato (preferencialmente o XML), por questões de compatibilidade.

Para definir o tipo de dados que a API deverá retornar a partir de uma requisição, o cliente deverá passar o cabeçalho **Accept** na requisição com o tipo de retorno que o mesmo deseja (**application/xml** para o tipo de retorno XML ou **application/json** para o tipo de retorno JSON). Caso o tipo de dado informado pelo cliente não seja suportado pela API, deve-se retornar o código de status HTTP **415 - Unsupported Media Type** ou **400 – Bad Request** e se o cabeçalho não for informado deve-se retornar o formato JSON por padrão.

Na resposta da requisição é necessário informar no cabeçalho **Content-Type** o tipo da resposta que a API está enviando.

2.5 CÓDIGOS DE STATUS HTTP

As respostas devem incluir, além da mensagem, o código de status HTTP de acordo com o tipo de resposta.

O uso correto de códigos de status nas respostas ajuda os desenvolvedores de aplicações, portanto, evite utilizar o status 200 para todos os tipos de resposta.

Além disso, em respostas com erro, sempre complemente o status com mensagens de retorno explicativas. Exemplo:

```
HTTP 400 Bad Request
{“msg_erro”:”o formato de mensagem solicitado não é suportado”}
```

Utilize os três grupos de códigos de resposta para indicar: sucesso (2xx), falha devido ao problema do lado do cliente (4xx) ou falha devido ao problema do lado do servidor (5xx). A seguir são listados os códigos HTTP mais comuns:

CÓDIGO*	DESCRIÇÃO
200 - OK	A chamada foi bem sucedida (**USO GERAL**).
201 – Created	Inclusão de recurso efetuada com sucesso (método POST).
204 – No Content	A consulta não retornou nenhum objeto.
206 - Partial Content	O conteúdo da resposta é paginado.
400 - Bad Request	A requisição é inválida ou mal formatada (**USO GERAL**).
401 - Unauthorized	As credenciais de acesso não foram informadas ou são inválidas.
403 - Forbidden	As credenciais são válidas, mas o usuário informado não tem acesso ao recurso.
404 - Not Found	O recurso acessado não existe (**USO GERAL**).
415 - Unsupported Media Type	O tipo de dado solicitado não é suportado pela API.
429 - Too Many Requests	O usuário atingiu o limite de requisições.
500 - Internal Server Error	Houve um erro interno do servidor ao processar a requisição. Consulte o status dos servidores (**USO GERAL**)

*Nomes e códigos seguem a RFC 7231 do IETF (https://tools.ietf.org/html/rfc7231)

Existe a possibilidade de utilização dos demais códigos de status, de acordo com o tipo de resposta, desde que a sua utilização esteja devidamente documentada.

Os códigos de status as serem utilizados devem ser definidos durante o design da API de acordo com a necessidade e as características das aplicações consumidoras.

2.6 LIMITES DE REQUISIÇÕES

As informações sobre os limites de requisições e contagem total definidas para uma API ou cliente devem estar disponíveis para os consumidores/clientes.

Caso um cliente extrapole o limite de requisições definido para uma determinada API, o servidor deve retornar o status HTTP **429 - Too Many Requests**. O servidor poderá retornar no header (**Retry-After**) o número de segundos que se deve esperar até realizar a próxima requisição.

2.7 HYPERMEDIA

A API poderá oferecer links de hypermedia para facilitar a descoberta de recursos e operações por parte dos clientes. Cada chamada para a API pode retornar no corpo da mensagem de resposta todos os possíveis estados da aplicação a partir do estado atual.

Para implementar, links pode-se utilizar as anotações de Link da **RFC5988**, por exemplo:

```
https://api.caixa/cadastro/v1/clientes/002

HTTP 200 OK
{
“cliente”:{"id":"002", "firstname":" Ana"},
“links”:[
  {“href”:“https://api.caixa/cadastro/v1/clientes/002/enderecos”,
“rel”:”endereços”}
  ...]
}
```

2.8 SEGURANÇA

2.8.1 CORS (CROSS-ORIGIN RESOURCE SHARING)

Uma política de mesma origem (**same-origin policy**) é uma restrição que tem como objetivo impedir que aplicações Web pertencentes a um determinado domínio façam requisições para um recurso em outro domínio.

Os navegadores web possuem essa restrição, portanto, aplicações Web ou Móveis que utilizam AJAX para acessar APIs Web de outros domínios não irão funcionar a menos que as APIs implementem o padrão CORS. O CORS (Cross-Origin Resource Sharing) é um padrão W3C que permite que o servidor flexibilize o acesso aos seus recursos oriundos de outras origens.

Durante o fluxo do CORS, a aplicação encaminha uma pré-requisição para a API utilizando método HTTP OPTIONS e informando a origem, os cabeçalhos e o método HTTP da requisição original. Para tal, são utilizados os cabeçalhos HTTP **Origin**, **Access-Control-Request-Headers** e **Access-Control-Request-Method**, respectivamente.

Do outro lado, o provedor da API deve tratar a requisição e responder com as origens, cabeçalhos e métodos autorizados utilizando os cabeçalhos HTTP **Access-Control-Allow-Origin**, **Access-Control-Allow-Headers** e **Access-Control-Allow-Methods**, respectivamente.

Caso todos os aspectos da requisição original sejam autorizados, a requisição é finalmente encaminhada para o provedor da API.

2.8.2 UTILIZAÇÃO DE SSL

Ao usar SSL, que é uma comunicação criptografada, simplificam-se os esforços de autenticação sendo possível trabalhar com tokens de acesso simples ao invés de ter que assinar ou criptografar as mensagens a cada solicitação à API.

É importante que os desenvolvedores de aplicações consumidoras das APIs estejam cientes das implicações de se utilizar SSL, como por exemplo, a necessidade de importação dos certificados do servidor de API para dentro da aplicação consumidora.

2.8.3 CONTROLE DE ACESSO

Caso a API seja de acesso restrito, deve-se utilizar mecanismos de segurança para controlar o acesso aos recursos.

O protocolo a ser utilizado para a segurança das API é o **OAuth2 (Open Authentication)**, que trabalha com tokens de acesso no formato **JWT (JSON Web Token)**.

Cada requisição deve conter no cabeçalho HTTP **Authorization** um token que foi concedido ao cliente por um provedor de identidade mediante autenticação prévia. Exemplo:

```
Authorization Bearer YHD678983GHJGKUD.GDUKHDIYHDTUYTKHGA.JASTYUDAJDG
```

O provedor do recurso deve ser capaz de validar os tokens internamente (assinatura, data de expiração, emissor, aplicação cliente e usuário) ou pode submeter o token recebido para ser validado no provedor de identidade que o emitiu.

Caso o cliente faça uma requisição para um recurso protegido sem o token ou com um token inválido, o servidor deve retornar o status HTTP **401 – Unauthorized**. Caso o token seja válido, porém o usuário informado não tem acesso ao recurso, o servidor deve retornar o status HTTP **403 – Forbidden**.

Como alternativa à utilização do protocolo oAuth2, poderá ser usado o tipo Basic. Nesse caso, cada requisição deve conter no cabeçalho HTTP Authorization o usuário e a senha. O usuário e a senha devem estar entre chaves, concatenados por “:” (dois pontos) e convertidos para Base64, conforme abaixo:

```
Authorization Basic Base64({usuario}:{senha})
```

O uso de autenticação do tipo Basic não é recomendado e deve ser destinada apenas para identificação de usuário de serviço e nunca com usuário e senha de pessoas.

Outra forma de identificação das aplicações consumidoras das APIs é por meio de API Key, as quais são fornecidas e geridas por uma solução de API Management. Geralmente as API Keys trafegam dentro do cabeçalho HTTP Apikey.

3 DOCUMENTAÇÃO DAS APIS

A documentação das APIs deve ser fácil de encontrar, de acesso público e conter exemplos completos de utilização (requisição/resposta).

O padrão a ser adotado para documentar as APIs é o Swagger (Open API). Análogo ao WSDL e XSD, o padrão Swagger define uma notação para documentar APIs RESTful.

Um documento Swagger pode ser escrito no formato JSON ou YAML e a sua estrutura deve conter:

- Informações básicas da API, tais como, nome, descrição, responsáveis etc;
- O **host** (endereço do servidor da API) e o **basePath** (onde geralmente é informada a versão da API);
- O **scheme**, que informa qual o protocolo a API utiliza (http/https) e os formatos de mensagem suportados para requisição (**consumes**) e resposta (**produces**);
- Os recursos disponibilizados por meio da API (**path**), seus métodos (GET, POST, PUT, DELETE) e os parâmetros de entrada e saída;
- As definições dos objetos representados pelos recursos da API.

Detalhamento das especificações pode ser encontrado em http://swagger.io/specification e a sua leitura é extremamente recomendada.

Existem ferramentas que geram uma visão amigável e interativa dos recursos de uma API, a partir de um documento Swagger.

Algumas linguagens e IDEs também permitem a geração de documentos Swagger a partir do código fonte da API. Um exemplo é a especificação JSR 311 - JAX-RS (http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html), que permite ao desenvolvedor implementar APIs Web em Java de maneira fácil e ainda documenta-las no padrão Swagger.

Além do padrão Swagger, existem também os padrões WADL e RAML, o primeiro é escrito em XML e se assemelha ao WSDL e o segundo é escrito em YAML e se assemelha ao Swagger. É importante que se conheça esses padrões alternativos, pois soluções de API Management ou mesmo alguns clientes podem não suportar o padrão Swagger.

A seguir temos um exemplo de documento Swagger no formato YAML:

```
Info:
  title:  API de Loterias
  description: API para a aposta de loterias
  version: '1.0.0'
  termsOfService: https:\\api.caixa\terms
  contact:
    name: Fulano de Tal
    email: apicaixa@caixa.gov.br
host: api.caixa
basePath: loterias/v1
schemes:
  - https
consumes:
  - application/json
produces:
  - application/json
paths:
  /mega_sena:
      post:
           summary: Incluir uma aposta de mega-sena
      consumes:
        - application/json
      produces:
        - application/json
        - application/xml
      parameters:
        - in: body
          name: body
          schema:
            $ref: “#/definitions/mega-sena”
      responses:
        '200':
          description: Aposta incluída
        ‘400’:
          description: Parâmetros de aposta inválidos
definitions:
  mega-sena:
    type: object
    properties:
      qtde-numeros:
        type: int
      numero:
        type: int
```

4 REFERÊNCIAS

- Web API Design. Acesso em Março de 2017, disponível em: https://pages.apigee.com/rs/351-WXY-166/images/ebook-2013-03-wad.pdf
- API Facade Pattern. Acesso em Março de2017, disponível em: https://pages.apigee.com/rs/351-WXY-166/images/api-facade-pattern-ebook-2012-06.pdf
- A World About Microservice Architectures and SOA. Acesso em Março de 2017, disponível em: http://soacommunity.com/index.php/magazine/articles/236-articles-microservicesweir
- Projeto de API RESTful – OCTO Guia de Referência Rápida. Acesso em Abril de 2017, disponível em: http://blog.octo.com/wp-content/uploads/2015/12/RESTful-API-design-OCTO-Quick-Reference-Card-PT-2.3.pdf
- Projetando uma API REST. Acesso em Abril de 2017, disponível em : http://blog.octo.com/pt-br/projetando-uma-api-rest/
- Web Link. Acesso em Abril de 2017 disponível em: https://tools.ietf.org/html/rfc5988
- Swagger Specification. Acesso em Abril de 2017, disponível em: http://swagger.io/specification/
- Editor Swagger. Acesso em Abril de 2017, disponível em: http://editor.swagger.io
- REST API Tutorial. Acesso em Dezembro de 2017, disponível em: http://restapitutorial.com