CAIO CESAR ALVES BORGES
KENNETH GOTTSCHALK DE AZEVEDO

A IMPORTÂNCIA DE USAR PADRÕES DE DESIGN EM APIS REST

Artigo apresentado ao *Uni-FACEF Centro Universitário Municipal de Franca* para obtenção do título de pós-graduado(a) em Desenvolvimento de Aplicações Web e Móveis Escaláveis.

FRANCA
2021

RESUMO

Atualmente, a utilização de APIs (Interface de Programação de Aplicações) vem se tornando cada vez mais comum no contexto corporativo, o que não é de se estranhar, já que elas proporcionam uma vasta quantidade de benefícios envolvendo aspectos como praticidade, performance e segurança. Em meio a esta quantidade numerosa de benefícios, existe a possibilidade de integrar aplicações, no entanto, para que este feito se realize, é imprescindível que a aplicação solicitante de recursos siga uma documentação antes de usufruir das funcionalidades fornecidas pela API, e é neste momento que se deve enfatizar a importância do uso de padrões de design. O objetivo deste trabalho é, com base em pesquisas, apresentar os possíveis problemas causados pela falta de padrão e sugerir boas práticas de desenvolvimento usando como referência uma API feita pelos próprios autores em virtude dos conhecimentos adquiridos durante o curso, sempre enfatizando de que nada descrito tem a obrigação de ser seguido.

Palavras-chave: Padrões de Design. API REST. Integração de Aplicações.

Submissão: 15/03/2021

1 INTRODUÇÃO

A partir do momento que o desenvolvimento de APIs se destacou na área da tecnologia, o número de empresas que resolveram utilizá-las aumentou drasticamente, porém, muitas destas empresas não estabeleceram padrões em suas APIs, não tendo noção dos problemas que isso poderia acarretar no futuro. Esse tipo de situação também ocorre com frequência nos dias atuais.

Tendo em conta este cenário, foi feita uma pesquisa sobre quais deveriam ser os pontos mais relevantes a serem levados em consideração pelas APIs para que elas fossem consideradas de alta qualidade e de fácil entendimento, proporcionando assim, uma integração sem complicações entre as aplicações.

Com base nas buscas realizadas, foram descobertos diversos tipos de padrões válidos e concluiu-se que é fundamental que uma API utilize os verbos corretamente de acordo com cada ação, contenha URLs que estejam relacionadas às entidades, disponibilize a opção de paginar e ordenar registros, retorne os códigos de status corretos com base em cada tipo de situação, siga um padrão de nomenclatura e forneça uma documentação fácil de ser interpretada.

A API desenvolvida pelos autores engloba todos os aspectos abordados anteriormente, possuindo operações de busca, inclusão, alteração e exclusão de registros, tais como: desenvolvedores, habilidades e a junção das duas entidades. Os exemplos citados neste trabalho estão baseados nesta API e o link da sua documentação está nas referências.

Levando em consideração que este trabalho possui um tema de difícil compreensão, os autores visaram explicar e exemplificar cada um dos assuntos abordados na pesquisa de uma maneira fácil de entender, buscando agregar todos os tipos de leitores.

Se uma API for desenvolvida com base nas sugestões descritas neste trabalho, ela será considerada apta para realizar integrações sem enfrentar nenhum tipo de dificuldade.

2 API REST

Não se pode falar sobre padrões de design em APIs REST sem antes entender o significado de API e como ela é caracterizada como REST.

2.1 DEFINIÇÃO

A sigla API é a abreviatura de *Application Programming Interface* e se trata de uma interface que possibilita a comunicação entre duas ou mais aplicações distintas. Quando uma API precisa enviar dados através da rede, ela é caracterizada como um Web Service. Desse modo, todo Web Service é considerado uma API, porém, nem toda API é considerada um Web Service.

> *Os Web Services surgiram como consequência natural da utilização da Internet. Alguns consideram essa utilização massificada, como um processo que produz a evolução desse meio de comunicação entre pessoas, e também como grande rede de computadores, o que naturalmente levou à possibilidade de se escrever aplicações e disponibilizá-las ao público em grande escala. (ABINADER; LINS. 2006, p. 10)*

Já a sigla REST, que é a abreviatura de *Representational State Transfer*, se trata de um conjunto de padrões de arquitetura que devem ser seguidos para a criação de Web Services. RESTful é um termo atribuído à API que segue os padrões REST.

As APIs têm como principal função expor recursos na Web, sendo que cada recurso possui uma maneira diferente de ser adquirido. Para isso, o client (aplicação que solicita recursos) e o server (aplicação que fornece recursos) se comunicam através de mensagens HTTP compostas por URL, verbo, cabeçalho(s), corpo e código de status.

2.2 VERBOS

Um dos dados de requisição obrigatórios durante a chamada de API é o verbo (ou método). Ele representa a ação que será realizada e é essencial que seja utilizado da maneira correta. Os principais verbos são:

- GET — obtenção dos dados de um ou mais registros;
- POST — inclusão de um novo registro;
- PUT — alteração dos dados de um registro;
- PATCH — alteração dos dados de um registro (de forma parcial);
- DELETE — exclusão de um ou mais registros.

O verbo GET, ao contrário dos outros, é considerado seguro, pois não há modificação de recursos.

Há casos em que o verbo POST é utilizado para realizar operações lógicas ao invés de incluir novos registros, como, por exemplo, no retorno de um Webhook.

2.3 URL

A URL (*Uniform Resource Locator*) também é um dado de requisição obrigatório e, quando junta ao verbo, servem para identificar unicamente um recurso na Web. A sintaxe de uma URL é *protocolo://domínio:porta/caminho*, sendo:

- Protocolo — HTTP ou HTTPS;
- Domínio — nome do servidor Web ou endereço IP;
- Porta — número da porta utilizada pela aplicação;
- Caminho — nome e localização de determinado recurso.

As URLs também podem possuir variáveis de caminho e/ou parâmetros de requisição, que consistem em filtrar uma busca ou identificar um registro. Segue abaixo alguns exemplos de como realizar a junção do verbo com a URL (ressaltando que, nas rotas de inclusão e alteração, os dados de requisição do registro ficam no corpo da mensagem).

Figura 1 — Exemplo de rotas para buscar, incluir, alterar ou excluir registros.

Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

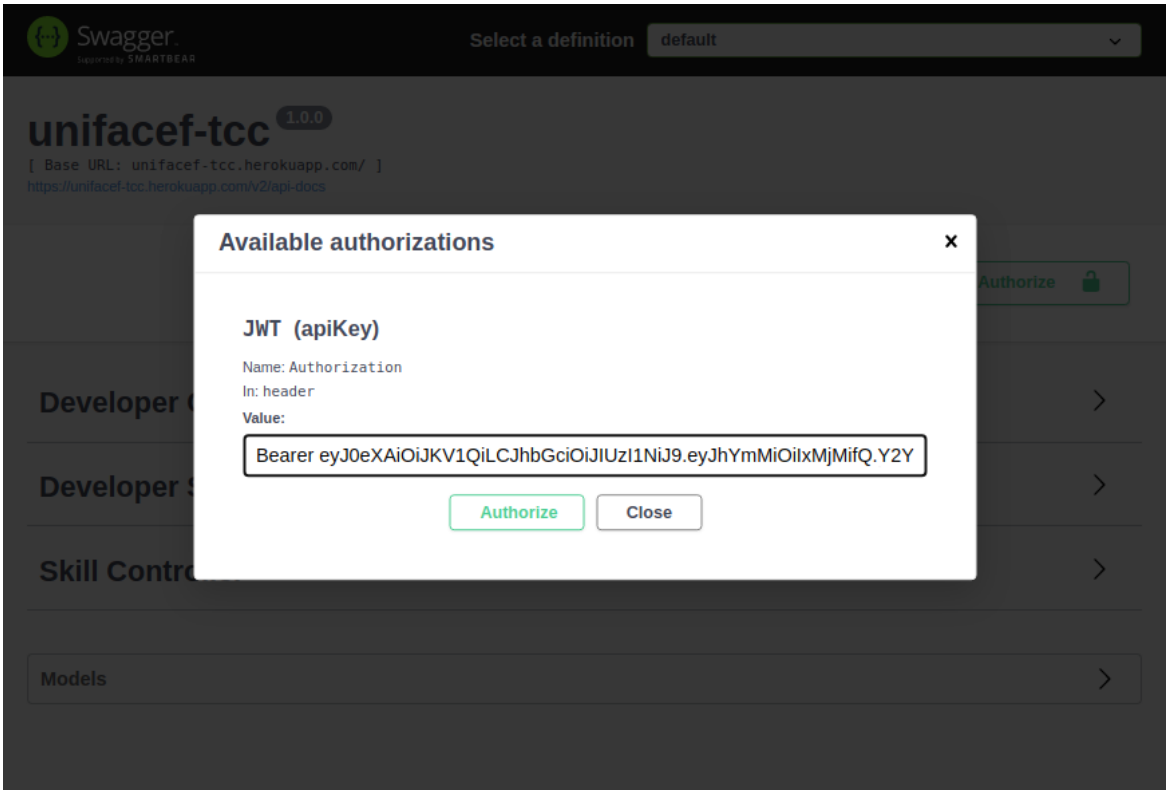
Existem desenvolvimentos que causam uma quebra no contrato, não sendo possível reaproveitar as rotas já existentes. Esse tipo de problema ocorre em situações como: alteração do verbo, alteração no caminho da URL, adição de dados de requisição obrigatórios, alteração de nome ou tipagem de um dado, remoção de dados de resposta e alteração no código de status.

Para resolver esse tipo de situação é necessário adicionar uma nova versão nas rotas que foram desenvolvidas (manter a *v1* e adicionar a *v2*, por exemplo) e adequar os clients para utilizá-las.

2.4 HEADERS

Os headers (cabeçalhos) são informações opcionais introduzidas nas requisições e nas respostas de uma chamada de API. Geralmente são informações distintas e não possuem relação com os dados da entidade que a URL se associa.

Figura 2 — Exemplo de adição de um token JWT no header das requisições.



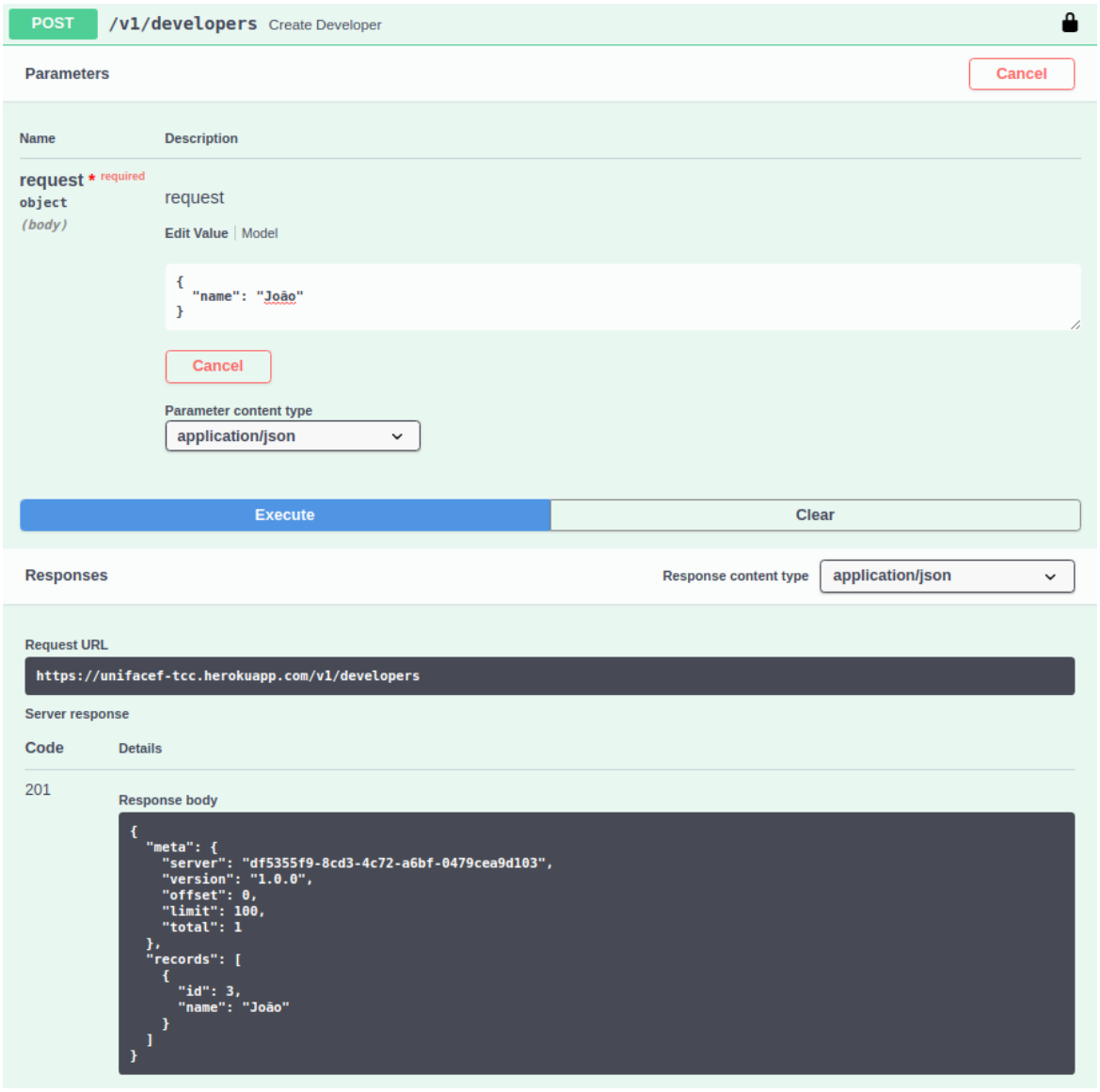
Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

Uma ótima maneira de se realizar autenticações é introduzindo chaves de API no cabeçalho da requisição, independentemente do tipo da autenticação. No caso da API desenvolvida pelos autores, foi-se utilizada a autenticação JWT e a chave da API está exposta no Swagger para facilitar os testes feitos pelos usuários (lembrando que não é uma boa prática expor dados sensíveis em uma documentação, sendo este, um caso de exceção).

2.5 BODY

O body (corpo da mensagem) é um conteúdo opcional introduzido nas requisições e nas respostas de uma chamada de API. Este conteúdo geralmente possui um formato de texto informado no cabeçalho, sendo que os mais utilizados são XML e JSON.

Figura 3 — Exemplo de conteúdo no corpo da mensagem.



Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

No exemplo acima, o corpo da mensagem de resposta é separado em *meta* (contendo dados de paginação, nome do servidor e versão da aplicação) e *records* (contendo os registros). Esse tipo de padrão facilita o desenvolvimento nas aplicações que forem consumir os recursos da API, pois todas as respostas possuem o mesmo formato e sempre haverá registros nas chamadas com sucesso.

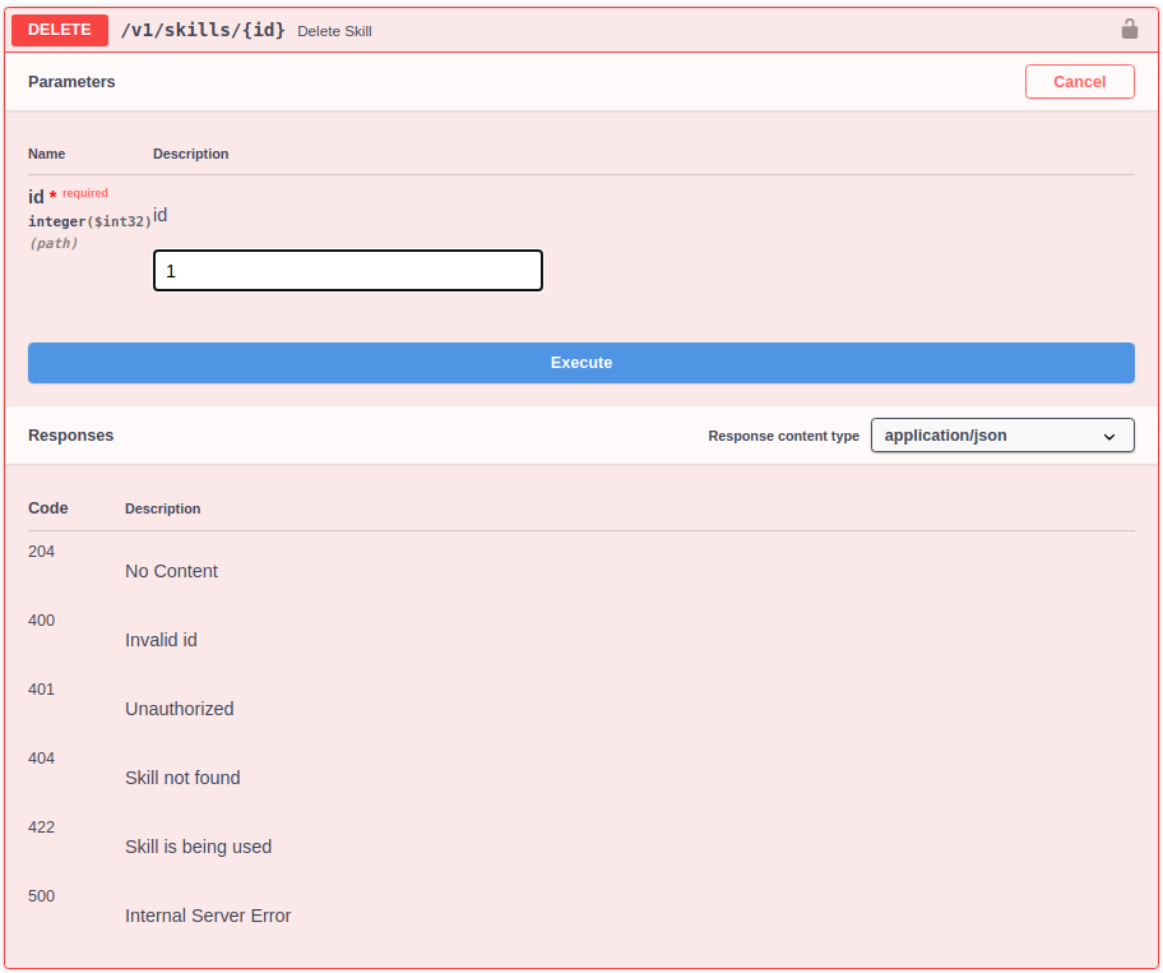
É aconselhável que o corpo das mensagens possua uma relação com a entidade que a rota se associa (em exceção dos casos de exclusão, pois não há conteúdo de resposta).

2.6 STATUS CODE

O status code (código de status) se trata de um número que identifica se a resposta de uma chamada de API foi processada com sucesso ou não. Eles são classificados em famílias:

- Família 100 (de 100 à 199) — informação;
- Família 200 (de 200 à 299) — sucesso;
- Família 300 (de 300 à 399) — redirecionamento;
- Família 400 (de 400 à 499) — erro por parte do client;
- Família 500 (de 500 à 599) — erro por parte do server.

Figura 4 — Exemplo dos possíveis códigos de status na exclusão de um registro.



Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

Na figura abaixo é possível se deparar com a quantidade imensa de códigos de status diferentes criados até o momento.

Figura 5 — Lista de todos os códigos de status até o momento.

1xx: HTTP Informational Codes	
100	Continue
101	Switching Protocols
102	Processing WebDAV
103	Checkpoint draft POST PUT
122	Request-URI too long IE7

2xx: HTTP Successful Codes	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information 1.1
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status WebDAV 4918
208	Already Reported WebDAV 5842
226	IM Used 3229 GET

3xx: HTTP Redirection Codes	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other 1.1
304	Not Modified
305	Use Proxy 1.1
306	Switch Proxy unused
307	Temporary Redirect 1.1
308	Permanent Redirect 7538

307 and **308** are similar to **302** and **301**, but the new request method after redirect must be the same, as on initial request.

4xx: HTTP Client Error Code	
400	Bad Request
401	Unauthorized
402	Payment Required res
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot 2324
422	Unprocessable Entity WebDAV 4918
423	Locked WebDAV 4918
424	Failed Dependency WebDAV 4918
425	Unordered Collection 3648
426	Upgrade Required 2817
428	Precondition Required draft
429	Too Many Requests draft
431	Request Header Fields Too Large draft
444	No Response nginx
449	Retry With MS
450	Blocked By Windows Parental Controls MS
451	Unavailable For Legal Reasons draft
499	Client Closed Request nginx

5xx: HTTP Server Error Codes	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates 2295
507	Insufficient Storage WebDAV 4918
508	Loop Detected WebDAV 5842
509	Bandwidth Limit Exceeded nostd
510	Not Extended 2774
511	Network Authentication Required draft
598	Network read timeout error nostd
599	Network connect timeout error nostd

HTTP Code Comments	
WebDAV	WebDAV extension
1.1	HTTP/1.1
GET, POST, PUT, POST	For these methods only
IE	IE extension
MS	MS extension
nginx	nginx extension
2518, 2817, 2295, 2774, 3229, 4918, 5842	RFC number
draft	Proposed draft
nostd	Non standard extension
res	Reserved for future use
unused	No more in use, deprecated

Wikipedia was used to produce all HTTP codes content: http://en.wikipedia.org/wiki/HTTP_status

Fonte: Http Status Codes Cheat Sheet – Konstantin Stepanov.

Implementar todos os códigos de status à risca é uma tarefa complexa e demorada. Para isso, os códigos considerados mais importantes foram selecionados e especificados por categoria.

Lista dos códigos de status de sucesso mais utilizados:

- 200 (OK) — utilizado em rotas não paginadas do tipo GET, em rotas do tipo PUT/PATCH e em rotas do tipo POST que realizam operações lógicas ao invés de inserir novos registros;
- 201 (CREATED) — utilizado em rotas do tipo POST que fazem inserção de novos registros;
- 202 (ACCEPTED) — utilizado em rotas que realizam processos de maneira assíncrona;

- 204 (NO CONTENT) — utilizado em rotas que não possuem conteúdo de resposta, como por exemplo, rotas do tipo DELETE;
- 206 (PARTIAL CONTENT) — utilizado em rotas que possuem paginação;
- 207 (MULTI-STATUS) — utilizado em rotas que processam múltiplos registros ao mesmo tempo, podendo existir registros processados com sucesso e registros processados com erro em uma mesma chamada.

Lista dos códigos de status de erro por parte do client mais utilizados:

- 400 (BAD REQUEST) — utilizado quando o client preenche algum dado de requisição incorretamente;
- 401 (UNAUTHORIZED) — utilizado quando os dados de autenticação informados pelo client são inexistentes ou inválidos;
- 403 (FORBIDDEN) — utilizado quando o client não possui permissão para realizar determinada ação;
- 404 (NOT FOUND) — utilizado quando determinado caminho/registro não é encontrado;
- 408 (TIMEOUT) — utilizado quando ocorre timeout durante o processamento dos registros;
- 409 (CONFLICT) — utilizado quando o client tenta criar um registro já existente ou processar algo já processado;
- 410 (GONE) — utilizado quando o client tenta processar um registro que foi expirado por algum motivo;
- 412 (PRECONDITION FAILED) — utilizado quando ocorre erro nas validações feitas antes do processamento dos registros;
- 422 (UNPROCESSABLE ENTITY) — utilizado quando o client informa dados de requisição válidos, porém, não processáveis;
- 429 (TOO MANY REQUESTS) — utilizado quando o servidor atingiu o limite máximo de processamento de registros.

Lista dos códigos de status de erro por parte do server mais utilizados:

- 500 (INTERNAL SERVER ERROR) — utilizado quando ocorre erros inesperados ou falta de tratativas durante o processamento dos registros;

- 502 (BAD GATEWAY) — utilizado quando uma dependência externa apresenta algum tipo de comportamento inesperado.

2.7 PAGINAÇÃO E ORDENAÇÃO

As rotas que retornam mais de um registro precisam ser paginadas pois, desta forma, será trafegado somente o necessário e tornará a API mais performática, oferecendo uma melhor experiência ao usuário final.

Uma das formas existentes de paginar registros é informar, via parâmetros de requisição, o número da página atual e a quantidade de registros por página. É desejável que o total de registros seja retornado pela API para que a aplicação solicitante saiba a quantidade de páginas existentes.

Figura 6 — Exemplo de uma busca de registros com paginação.

Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

Em relação à ordenação, é aconselhável que essa opção esteja do lado da API para evitar que haja desenvolvimento nas aplicações solicitantes.

Uma forma válida de ordenar registros é informando o nome do campo que será ordenado junto ao tipo de ordenação (crescente ou decrescente), por exemplo: *v1/developers?offset=0&limit=1&sort=+name*.

3 PADRÕES DE NOMENCLATURA

É essencial que haja um padrão de nomenclatura nas rotas e nos dados de requisição/resposta, pois isso facilita a interpretação da documentação da API. Há diversos tipos de padrões, porém, é imprescindível que apenas um deles seja utilizado.

Exemplo de padrões que podem ser utilizados nas rotas de uma API:

TIPO	**EXEMPLO**
Plural	/v1/products/
Singular	/v1/product/
Camel Case	/v1/marketplaceProducts/
Snake Case	/v1/marketplace_products/
Spinal Case	/v1/marketplace-products/

No caso dos dados de requisição e de resposta, é aconselhável utilizar o plural para as listas e o singular para o restante. O camel case, snake case e spinal case também são exemplos de padrões que podem ser aplicados a estes dados.

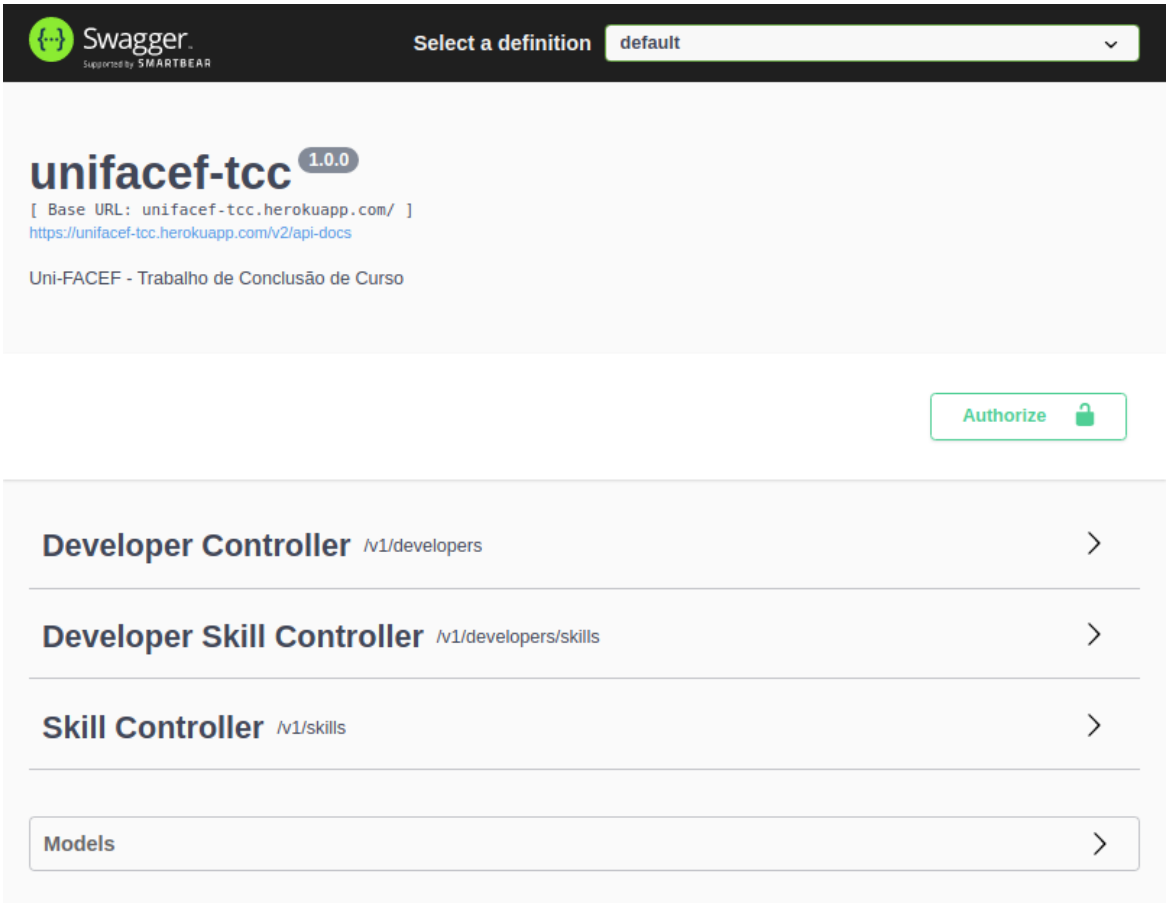
Vale salientar de que é importante utilizar apenas uma língua na API (não misturar português e inglês, por exemplo).

4 DOCUMENTAÇÃO

Após desenvolver uma API, é necessário expor as suas funcionalidades aos clients e, para isso, uma documentação precisa ser criada. Uma ótima alternativa é a utilização do Swagger.

“A principal vantagem do Swagger é que o mesmo possui uma dependência para as principais linguagens de programação do mercado, e funciona basicamente lendo as anotações do código, o que facilita muito que a cada alteração no código, sua documentação permaneça atualizada” (SILVA, 2018).

Figura 7 — Exemplo de documentação via Swagger (versão 3.0.0).



Fonte: Documentação da API do Trabalho de Conclusão de Curso – Os autores.

Uma outra grande vantagem de se utilizar o Swagger é a possibilidade de chamar as APIs através dos clients gerados por ele, facilitando a utilização das funcionalidades e a realização dos testes.

5 CONCLUSÃO

Se uma API contém todas as especificações citadas neste trabalho, ela é considerada de padrão e de alta qualidade. O desenvolvimento de APIs em si é um assunto muito amplo, não sendo possível relatar e padronizar todos os casos. Ocorrerá situações onde uma API deverá ser desenvolvida de uma maneira diferente para atender aos requisitos de uma determinada demanda, não sendo possível seguir à risca todas as boas práticas aqui descritas. Mas sempre haverá uma solução para cada tipo de situação e, com uma API bem documentada, não haverá problemas com a integração entre as aplicações.

REFERÊNCIAS

ABINADER, Jorge Abílio; LINS, Rafael Dueire. *Web Services em Java*. Rio de Janeiro: Brasport, 2006.

SILVA, William da. *Design de API Rest*. Disponível em: https://medium.com/@wssilva.willian/design-de-api-rest-9807a5b16c9f. Acesso em: 20 fev. 2021.

STEPANOV, Konstantin. *HTTP Status Codes Cheat Sheet*. Disponível em: https://cheatography.com/kstep/cheat-sheets/http-status-codes/. Acesso em: 28 fev. 2021.

BORGES, Caio Cesar Alves; AZEVEDO, Kenneth Gottschalk de. *Documentação da API do Trabalho de Conclusão de Curso*. Disponível em: https://unifacef-tcc.herokuapp.com/swagger-ui/. Acesso em: 01 mar. 2021.