Aprimorando o uso de Links em API Contract

Richard Lucas Lima Auzier[1], **Ailton Santos Filho**[1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação (IComp) - Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brazil

{rlla,assf,efeitosa}@icomp.ufam.edu.br

Abstract. *Web REST APIs have been widely used in applications that demand web communication. Although several specifications have emerged for modeling and documenting these APIs (OpenAPI and Swagger), none describe the relationships between an API's operations, which would help developers more quickly create clients for these APIs and detect security holes. Thus, the objective of this research is to develop a heuristic, extending those existing in the literature, for the automatic generation of OpenAPI Links, a mechanism defined in the OpenAPI 3.X specification, to allow the description of relationships between API operations, for example, the precedence between operations (control flow) and how the result of an operation can be used as input for a subsequent operation (data flow).*

Resumo. *As APIs Web REST vêm sendo largamente utilizadas em aplicações que demandam comunicação Web. Embora existam especificações para modelar e documentar essas APIs (OpenAPI e Swagger) nenhuma descreve os relacionamentos entre operações de uma API, o que auxiliaria desenvolvedores a criar mais rapidamente clientes para essas APIs e a detectar falhas de segurança. Dessa forma, o objetivo desta pesquisa é desenvolver uma heurística, estendendo as existentes na literatura, para a geração automática de Links OpenAPI, um mecanismo definido na especificação OpenAPI 3.X, a fim de permitir a descrição de relacionamentos entre operações de uma API, como por exemplo, a precedência entre operações (control flow) e como o resultado de uma operação pode ser utilizado com entrada de uma operação seguinte (data flow).*

1. Introdução

As Web APIs (*Application Programming Interface*) vêm sendo cada vez mais utilizadas por oferecem, principalmente, redução no tempo e custo de desenvolvimento de aplicações web e móveis, bem como facilidade na integração com soluções de terceiros. Segundo relatório da Akamai, 83% do tráfego de rede monitorado por eles, em 2018, era oriundo desse tipo de APIs (Akamai 2019).

Uma vez que Web APIs enviam e recebem mensagens através do protocolo HTTP (*Hypertext Transfer Protocol*), sua comunicação é feita por meio de requisições à *endpoints* - o servidor ou o serviço disponibilizado - acessados através de uma URI (*Uniform Resource Identifier*), mais precisamente uma URL (*Uniform Resource Locator*). O exemplo ilustra uma requisição à um *endpoint* de uma das APIs do Spotify[1], que busca informações de vários artistas baseado nos seus identificadores.

[1]https://developer.spotify.com/documentation/web-api/

GET https://api.spotify.com/artists

No exemplo, o método HTTP **GET** (também chamado de operação no contexto da OpenAPI) é usado para recuperar recursos de um serviço, no caso informações sobre artistas. A primeira parte da URI (**https://api.spotify.com/**), chamada de *caminho base*, representa o servidor onde a API está localizada. Por fim, tem-se o *endpoint* em si (**/artists**). A requisição retorna, como resposta, um objeto chamado "artistas", cujo valor é um vetor contendo informações sobre o gênero, popularidade, imagens e outros dados sobre um determinado grupo de artistas.

Um aspecto importante sobre as Web APIs é sua especificação, que descreve *endpoints*, respostas e outras informações relevantes. Existem diferentes estratégias e linguagens para descrevê-las, mas a especificação OpenAPI é o formato de especificação de API mais usado. Segundo relatório da SmartBear, a OpenAPI representa 82% das APIs disponíveis em 2020 (SmartBear 2020).

A especificação OpenAPI (OAS) é um padrão de descrição agnóstico, derivado da especificação Swagger, criado inicialmente para descrever APIs no estilo REST (Fielding 2000). A especificação descreve informações sobre a API, sobre o fornecedor, termos de uso, endereço dos servidores e caminho para acesso aos recursos, formatos e requisitos para requisições e respostas, dentre outras (OpenAPI Initiative 2021). Uma especificação OAS pode ser descrita nos formatos JSON (*JavaScript Object Notation*) ou YAML (*YAML Ain't Markup Language*), seguindo a notação de objetos da linguagem JavaScript. Vale destacar que a partir da versão 3.1, a OAS permite descrever qualquer API que use o HTTP como seu protocolo de troca de mensagens (Ratovsky 2020).

Embora a especificação de Web APIs em OAS permita enumerar todas as operações fornecidas, ou seja, as combinações de *endpoint* de recurso e método, ela ainda carece da descrição de quais são as sequências significativas de interações da API que podem ser seguidas para atingir seus objetivos. Na versão 3.0 da OAS, foi adicionado um novo objeto chamado *links*, que tem a função de descrever como vários valores de parâmetros retornados por uma operação podem ser usados como entrada para outras operações (ErnstFriedman 2017). O objeto *link* cria uma relação entre uma operação do *endpoint A* e uma operação do *endpoint B*. A operação do *endpoint* B faz uso de algum valor que a operação do *endpoint A* produziu. Contudo, o uso do objeto *link* é opcional.

Em um estudo recente (Kus et al. 2020), os autores perceberam que, ao avaliar especificações de APIs reais, é dada pouca importância ao objeto *link* nas especificações. Apenas 3 especificações de API usando *link* foram encontradas em mais de 1.500 especificações analisadas do diretório APIs.guru[2]. Como resposta, os autores propuseram uma heurística para identificar, em especificações OAS, onde o objeto *link* poderia e deveria ser usado. Entretanto, a heurística foi testada apenas com pares de *endpoints*, *endpoint* A e B, que possuam o método HTTP GET.

Neste sentido, este artigo visa o aprimoramento da heurística proposta por (Kus et al. 2020) para identificação de objetos *link* em especificações OAS. A ideia é aperfeiçoar a heurística para identificação de relacionamentos (objeto *link*) entre diferentes métodos HTTP do lado do *endpoint* B, isto é, a ferramenta permanece identificando um *endpoint A* caso possua um método GET, já o *endpoint* B com métodos POST E

[2]https://apis.guru/

DELETE, além do GET, passam a ser identificados pela ferramenta.

2. OpenAPI e objeto Link

As Web APIs precisam de uma especificação completa e legível para permitir que seus consumidores possam aprender como usá-la e o que esperar de cada ação. A especificação OpenAPI, conhecida como OAS, é uma especificação aberta, desenvolvida em conjunto com a comunidade, com objetivo de desenvolver um padrão legível para humanos e computadores de descrição de Web APIs (também chamadas de *HTTP APIs*). Ela permite que um consumidor, ou usuário da API, interaja com o servidor remoto com uma quantidade mínima de lógica de implementação e sem a necessidade de outro tipo de documentação, ou acesso ao código fonte da API (OpenAPI Initiative 2021).

Um *OpenAPI Document* é um documento independente, ou grupo de vários documentos, que seguem a especificação OpenAPI e pode descrever uma API. Dentre os diversos componentes apresentados pela OpenAPI, o objeto *link* possui um papel importante na hora de tornar a definição da API mais intuitiva e natural. Após ler uma especificação de determinada API, é possível notar que certas operações podem produzir informações que são úteis para outras operações.

Por exemplo, um *endpoint* possui uma operação que cria um usuário e como resposta de sucesso retorna o *id* do usuário criado. Esse mesmo *id* é requerido por outra operação, como parâmetro, que modifica os dados de um usuário, ou seja, ela só modifica um usuário que já foi criado pela operação que cria um usuário. Essa operação é chamada de operação alvo, podendo ser do mesmo *endpoint* ou não, e a operação que possui a informação relevante pode ser chamada de operação de partida. Em outras palavras, caso um endpoint faça uso de um valor gerado por outro endpoint, então, existe uma relação entre esses endpoints. Caso essa relação pudesse ser modelada seria possível compreender como a API, em questão, se comporta. O objeto *link* é o responsável por sanar essa necessidade, refletindo quais são as possíveis interações entre os *endpoints*.

Para isso, o objeto *link* precisa saber de algumas informações: qual é a operação alvo? qual é o parâmetro da operação alvo que receberá o valor da resposta? Os campos responsáveis por indicar a operação alvo são chamados de *operationId* e *operationRef*, usados para indicar uma operação que está no documento e para indicar uma operação externa, respectivamente. Os dois campos não podem ser usados ao mesmo tempo. O campo responsável por indicar o parâmetro da operação alvo é chamado de *parameters*. Ele faz um mapeamento do nome do parâmetro para uma expressão ABNF (*Augmented Backus–Naur Form*), que expressa o valor literal que o parâmetro irá receber. O campo *description* explica o propósito do objeto *link* e descreve suas características gerais.

De acordo com a OAS, o objeto *link* é definido ou referenciado dentro do objeto *Response*, que descreve uma única resposta de uma operação API (Swagger 2021). Uma resposta para o objeto *Response* são os códigos de status das respostas HTTP que indicam se uma requisição HTTP foi corretamente concluída. O objeto *Response*, por sua vez, está dentro do objeto *Responses*, que é um *container* de respostas (códigos HTTP) esperadas para determinada operação. Então, é possível ter *links* para diferentes tipos de respostas. A Figura 1 ilustra a definição/referenciação do objeto *link* (campo link).

No exemplo da Figura 1, a operação (*createUser*, à esquerda) cria um usuário e retorna um *id*. Esse *id* é necessário para uma operação (*getUser*, à direita) que busca por

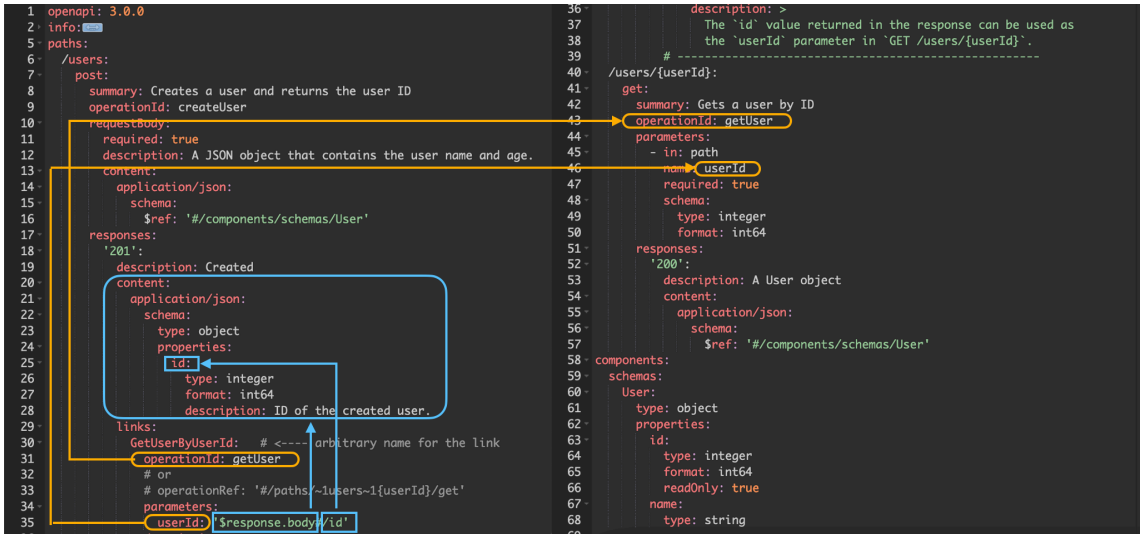

Figura 1. Definição do objeto link.

um usuário pelo seu *id*. Assim, no exemplo, o *link* foi nomeado como *GetUserByUserId* e possui os campos *operationId* e *parameters*, que foram explicados anteriormente. A existência do *link* não implica que a operação alvo irá ser capaz de invocá-lo com sucesso, pelo contrário, ele irá expressar uma relação conhecida e um mecanismo de travessia entre respostas e outras operações (OpenAPI Initiative 2021).

A primeira seta laranja destaca o campo *operationId*, informando que a operação *getUser* é a operação alvo. A segunda seta laranja destaca o parâmetro (*userId*) da operação alvo que receberá o valor da operação de partida. Este parâmetro está associado a uma expressão ABFN: *$response.body#/id*.

O *link* pode ser adicionado ou referenciado, mas os dois são feitos no escopo do *endpoint* (*endpoint A*) que está oferecendo alguma informação. Adicionar um objeto Link é especificar cada campo necessário, citado anteriormente. Referenciar um objeto *link* é especificar o endereço que a definição se encontra. O endereço dos objetos referenciáveis se encontram no objeto *Components*. Esse objeto serve para armazenar definições de outros objetos que são utilizados com frequência no decorrer do documento OpenAPI. Na Figura 1, o *link* foi adicionado de forma manual com os campos necessários sendo especificados. Um exemplo de referenciação do objeto *link* seria *$ref: "/components/links/categoriesFollowersGET"*. A definição do objeto *link categoriesFollowersGET* se encontra dentro do objeto *links* que está dentro do objeto *Components*, dessa forma, sempre que for necessário utilizar esse *link*, basta utilizar a localização da sua definição.

A forma de especificar quais dados serão passados para o parâmetro *userId* é ilustrado através das duas setas azuis. A primeira seta azul, que destaca o *$response.body*, está apontando para o corpo da resposta. Já a segunda seta azul está especificando a informação relevante que será usada pela operação *getUser* (o *id*). Esse exemplo serve para ilustrar, de forma geral, como funciona a relação de respostas para outras operações. É importante notar que a operação de partida (POST) pertence ao *endpoint /users*, a operação alvo (GET) pertence ao *endpoint* chamado */users/userId* e essa relação existe somente se o código de resposta HTPP (do *endpoint* A) for 200.

3. Heurística do Link Generator

No trabalho de (Kus et al. 2020), os autores decidiram investigar o uso do objeto *link* e elaboraram um gerador automático, permitindo adicioná-los à especificação OpenAPI 3.0, quando necessário. O *Link Generator* recebe um documento OpenAPI como entrada e como saída retorna o mesmo documento com os *links* adicionados. A ferramenta proposta pelos autores adota algumas heurísticas, que funcionam como critérios para separar e verificar quais *endpoints* possuem operações com a necessidade de um objeto *link*. As heurísticas podem ser divididas em dois grupos: um que separa os possíveis *endpoints* para o recebimento do objeto *link* e outro que valida os parâmetros desses *endpoints*.

O primeiro grupo está interessado em realizar uma filtragem em todos os *endpoints* do documento. A filtragem começa verificando quais *endpoints* possuem uma operação (método) GET e pelo menos uma resposta de sucesso HTTP. Após isso, é feito um tratamento, que consiste em formar pares de *endpoints* com todos eles sendo testados entre si. Para esse par ser formado, é necessário que o *endpoint* B tenha em seu início de nome o *endpoint* A. Quando isso acontece, dize-se que existe uma relação hierárquica entre esses *endpoints*. Essa relação hierárquica expressa a dependência do *endpoint* B com o *endpoint* A, ou seja, existe um valor gerado pelo *endpoint* A que é utilizado pelo *endpoint* B. A Figura 2 ilustra essa relação hierárquica.

Figura 2. Relação hierárquica entre os endpoints.

O *endpoint* A, na Figura 2, se chama */categories/categoriesID*. Ele forma pares com os *endpoints* que estão recebendo a seta. Por exemplo, o *endpoint /categories/categoryID/contributors* é um dos *endpoints* B que possui em seu inicio de nome o *endpoint* A (no caso, */categories/categoriesID)*.

Com esses pares selecionados, o segundo grupo de heurísticas é usado para verificar quais desses pares possuem operações com um parâmetro de mesmo nome e mesmo esquema. Se houver, esse parâmetro têm seu valor considerado igual entre diferentes operações. Ou seja, o parâmetro exigido pelo *endpoint* B precisa ser um subconjunto de todos os parâmetros do *endpoint* A. Na Figura 2, é possível notar que o parâmetro *categoriesID* do *endpoint* A é requirido pelos outros *endpoints*. Esse parâmetro precisa ter o nome igual entre o *endpoint* A e B, tal qual os seus esquemas. O esquema de um parâmetro é a sua estrutura e o tipo do valor daquele parâmetro (OpenAPI Initiative 2021). Por exemplo, se o parâmetro for um *id*, provavelmente o tipo desse parâmetro é um inteiro. Com estas condições satisfeitas, o conjunto final de *endpoints* pode receber o objeto *link*.

3.1. Limitações da ferramenta

Contudo, a ferramenta *Link Generator* possui suas limitações:

1. A solução de geração de *links* atua somente em métodos GET-GET (tanto o *endpoint* A quanto o *endpoint* B precisam conter um método GET). O motivo apresentado pelos autores é somente por questões de simplicidade da sua solução.
2. Falta de validação das heurísticas propostas em outros métodos.
3. A ferramenta se baseia em heurísticas morfológicas, ou seja, um *endpoint* B só tem uma relação com o *endpoint* A caso o nome do *endpoint* A faça parte do nome do *endpoint* B. *Endpoints* que não obedecem a essa regra são descartados.
4. Parâmetros com referência externa (ao documento) são descartados, por mais que aquele parâmetro seja válido.

4. Proposta

Esta pesquisa visa aperfeiçoar a primeira limitação do *Link Generator*, que opera somente com *endpoints* (A e B) que possuam o método GET. O *endpoint* A, da Figura 2, possui o método, assim como os *endpoints* que fazem par com ele. O aperfeiçoamento é direcionado para o *endpoint* B, permitindo que a ferramenta reconheça, além do método GET, os métodos POST e DELETE. O *endpoint* A continua com a necessidade de possuir um método GET. A Figura 3 ilustra a nova situação.

Figura 3. Geração de *Link* de GET para métodos GET, POST e DELETE.

Esse ajuste continua permitindo que o *endpoint* B (*categories/categoryID/followers*) seja identificado pelo primeiro grupo de heurística. Entretanto, agora, ele é aceito através dos três métodos (GET, POST, DELETE). Com essa melhoria, a ferramenta irá criar um *link* para cada operação do *endpoint* B. Vale destacar que o *link* pode ser adicionado se houver a ocorrência de apenas um, ou dois e ou os três métodos (GET, DELETE e POST).

4.1. Limitações da Proposta

Apesar da ferramenta modificada aumentar o número de *links* adicionados, ela permanece dependente de uma heurística baseada na morfologia dos nomes dos *endpoints*. Pelo fato do *Link Generator* avaliar somente *endpoints* B, que contenham o parâmetro já fornecido pelo *endpoint* A, a ferramenta perde relações que não são tão visíveis assim. Por exemplo, um *endpoint /users/*, com uma operação POST, cria um usuário e devolve um *id* que pode ser utilizado pela operação */users/id*, que busca um usuário. As heurísticas do *Link Generator* não identificam essa relação, pois não é possível encontrar o *id* no *endpoint* A (*L/users/*).

A ferramenta continua não aceitando parâmetros com referências externa, o que não permite que a especificação possa interagir com outros documentos OpenAPI que poderiam conter informações importantes para ela.

5. Resultados

A expansão do primeiro grupo de heurísticas trouxe um aumento no número de *links* adicionados. Em um teste com duas especificações parciais em OpenAPI (*openapi-*

reqbaz.json[3] e *spotify.json*[4]), a nova versão da ferramenta aumentou o número de links existentes na especificação. Antes da melhoria da ferramenta, a especificação *openapi-reqbaz.json* recebia 23 *links* e a especificação *spotify.json* recebia 19 *links*. Após a aplicação da ferramenta melhorada, o número de *links* subiu para 37 e 32, respectivamente.

Utilizando os *endpoints* (que foram retirados da especificação *openapi-reqbaz.json*) da Figura 2 como exemplo, pode-se observar o comportamento da ferramenta original. Ela adiciona somente um *link* do *endpoint* A (*/categories/categoryID*) ao *endpoint* B (*/categories/categoryID/followers)* (Figura 4).

```
"followers": {
  "description": "Automatically generated link definition",
  "operationId": "getFollowersForCategory",
  "parameters": {
    "categoryId": "$request.path.categoryId"
  }
},
```

Figura 4. Exemplo de *link* adicionado pela ferramenta original.

O *link* se chama *followers* e a operação alvo é uma operação GET (*/categories/categoryID/followers)*. Entretanto, como visto na Figura 3, o *endpoint* alvo, em questão, possui as operações GET, POST e DELETE. Com a melhoria, foram obtidos a adição de mais dois links, totalizando três links do *endpoint* A para o *endpoint* B. A Figura 5 ilustra os novos *links* adicionados.

```
categoriesFollowersGET:
  description:                    "Automatically generated link definition"
  operationId:                    "getFollowersForCategory"
  parameters:
    categoryId:                   "$request.path.categoryId"
categoriesFollowersPOST:
  description:                    "Automatically generated link definition"
  operationId:                    "followCategory"
  parameters:
    categoryId:                   "$request.path.categoryId"
categoriesFollowersDELETE:
  description:                    "Automatically generated link definition"
  operationId:                    "unfollowCategory"
  parameters:
    categoryId:                   "$request.path.categoryId"
```

Figura 5. *Links* adicionados pela ferramenta modificada.

Esses três *links* fazem parte do mesmo *endpoint*, mas pertencem a diferentes operações. Para cada *link*, o *operationId* muda, mas o parâmetro *categoryID* é o mesmo, pois é a informação relevante que *endpoint* A (*/categories/categoryID*) tem para compartilhar com o restante dos *endpoints*. A ferramenta original nomeava os links com o último nome do *endpoint*. O nome do *link* na Figura 4 é *followers* porque esse *link* pertence ao *endpoint /categories/categoryID/followers*. Para tornar o nome do *link* mais legível, a ferramenta modificada utiliza o primeiro e ultimo nome do *endpoint* mais o nome da operação. Por exemplo, *categoriesFollowersGET* é da operação GET e *categoriesFollowersPOST* é da operação POST.

[3]https://github.com/rwth-acis/openapi-link-generator/blob/master/test/fixtures/openapi-reqbaz.json
[4]https://github.com/APIs-guru/openapi-directory/tree/main/APIs/spotify.com/2021.7.20

6. Conclusões

Apesar do amplo uso das APIs Web, existe uma lacuna quanto a modelagem de relacionamentos entre suas operações e as interações entre as partes envolvidas. Em parte, porque trata-se de um processo manual que consome tempo dos especialistas. Esse tipo de informação traria benefícios aos desenvolvedores de software e profissionais de segurança, quanto ao entendimento do funcionamento da API e detecção de falhas de segurança (Ivanchikj and Pautasso 2020).

Dessa forma, com o objetivo de reduzir a carga de trabalho manual envolvida nesse processo, neste trabalho os autores aperfeiçoaram uma heurística para a criação automatizada de *links* OpenAPI. A fim de demonstrar a eficácia do método proposto, ele foi implementado como uma extensão da ferramenta apresentada em (Kus et al. 2020). Após testes com APIs comerciais, verificou-se que utilizando a heurística proposta, foi possível criar mais *links* do que era originalmente possível. Como trabalhos futuros, destaca-se: (i) a extensão da heurística proposta para outros métodos HTTP, como *PUT*; (ii) testes com um número maior de APIs reais; (iii) propor uma heurística que não dependa da morfologia dos *endpoints*, o que aumentaria o número de relações que poderiam ser modeladas entre os *endpoints*; e (v) acoplar a ferramenta uma funcionalidade de visualizar mais amigável (através de mapa ou diagrama).

7. Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001

Referências

[Akamai 2019] Akamai (2019). State of the internet / security: Retail attacks and api traffic. encurtador.com.br/coGW5. Acessado em: 15-10-2020.

[ErnstFriedman 2017] ErnstFriedman, J. (2017). A new year, a new specification. encurtador.com.br/jFHQZ. Acessado em: 05-03-2021.

[Fielding 2000] Fielding, R. T. (2000). Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*.

[Ivanchikj and Pautasso 2020] Ivanchikj, A. and Pautasso, C. (2020). Modeling Microservice Conversations with RESTalk. In *Microservices*, pages 129–146. Springer International Publishing.

[Kus et al. 2020] Kus, D. A., Koren, I., and Klamma, R. (2020). A link generator for increasing the utility of openapi-to-graphql translations. In *WWW2020 Developer Track*. ACM.

[OpenAPI Initiative 2021] OpenAPI Initiative (2021). The openapi specification: a broadly adopted industry standard for describing modern apis. https://www.openapis.org/. Acessado em: 05-03-2021.

[Ratovsky 2020] Ratovsky, R. (2020). Oas 3.1.0-rc0 released! encurtador.com.br/kqvNT. Acessado em: 05-03-2021.

[SmartBear 2020] SmartBear (2020). The state of api 2020 report. encurtador.com.br/ltyFQ. Acessado em : 14-03-2020.

[Swagger 2021] Swagger (2021). Links. https://swagger.io/docs/specification/links/. Acessado em: 05-03-2021.