Design de API Rest

Entenda uma série de boas práticas para sua API Rest

Willian da Silva Sep 9, 2018 · 10 min read

A sociedade evoluiu, a tecnologia evoluiu, por consequência a forma que nos comunicamos também mudou, isso também pode e deve ser aplicado a tecnologia.

Hoje API's são a base para qualquer negócio informatizado, portanto são a base para qualquer empresa, por isso vamos abordar abaixo conceitos básicos sobre o padrão Rest, sobre o protocolo HTTP e uma série de boas praticas visando a construção de uma API Rest de alto nível.

Rest e Restful

Representational State Transfer (Transferencia de Estado Representacional) ou Rest, representa uma série de princípios definidos pela World Wide Web, visando a padronização de rotas, requisições e comunicações sem estado, o próprio protocolo HTTP, foi baseado nestas regras e se propõe a resolver todos estes problemas.

Restful é o termo atribuído ha uma API que possui a inteligência de aplicar o padrão Rest.

Protocolo HTTP

O protocolo HTTP é a base de tudo, as requisições chegam e saem através dele, os padrões são baseados em sua estrutura, e seus códigos de resposta, por esse motivo, todas as nossas API's devem respeita-lo e aplicar suas melhores práticas, portanto segue abaixo algumas delas:

Verbos do protocolo HTTP

Existem 9 diferentes verbos do protocolo HTTP, vamos conhecer abaixo para que server cada um:

Verbos Protocolo HTTP	
Verbo	Objetivo
CONNECT	Utilizado para abrir uma comunicação bidirecional com determinado recurso
DELETE	Utilizado para deletar determinado domínio
GET	Utilizado para recuperar os dados de um determinado dominio
HEAD	Basicamente faz o mesmo que o verbo GET, porem na sua resposta só é retornado o cabeçalho da requisição
OPTIONS	Utilizado para recuperar as possiveis opções de requisições sobre um determinado domínio
PATCH / MERGE	Utilizado para atualizar parcialmente os dados de um determinado domínio
POST	Utilizado para criar dados de um determinado dominio ou realizar operações lógicas (Por exemplo um calculo)
PUT	Utilizado para atualizar dados de um determinado dominio
TRACE	Utilizado para enviar mensagens por todo o caminho entre a origem e o destino de uma requisição, provendo um grande mecanismo de debug

Verbos protocolo HTTP

A maioria das Api's utilizam apenas os verbos GET, POST, PUT e DELETE, porem isso não é uma regra, se fizer sentido para sua aplicação, utilize-as.

Padronização das rotas

Reduza a necessidade de documentações extensas, utilize paths legíveis ao olho humano, existem diversos tipos de padrões para as rotas, os mais utilizados estão exemplificados abaixo:

Roteamento	
Path	Tipo
Plural	/api/products/
Singular	/api/product/
Camel Case	/api/institutionalProducts

Camel Case	/api/institutionalProducts
Snack Case	/api/institutional_products
Spinal Case	/api/institutional-products

O ideal é que seja definido um padrão único para ser aplicado em todas as Api's da sua empresa, minha preferencia pessoal é utilizar o padrão no plural, portanto todos os exemplos deste post estarão seguindo esse padrão, segue abaixo um exemplo pratico cobrindo os principais verbos do protocolo HTTP:

Roteamento		
Requisição	Path	Objetivo
GET	/api/products	Consultar todos produtos
GET	/api/products/4	Consultar produto específico
POST	/api/products	Criar um produto
PUT	/api/products/7	Atualizar um produto específico
DELETE	/api/products/2012	Deletar um produto específico

Exemplo de roteamento de API no plural

Verbos do protocolo HTTP ao invés de operações

O exemplo a seguir demonstra rotas que possuem operações em seu nome, esse tipo de exemplo deve ser evitado

Roteamento		
Requisição	Path	Objetivo
GET	/api/products/consultar	Consultar todos produtos
POST	/api/products/cadastrar	Cadastra um novo produto
PUT	/api/products/4/alterar	Altera um produto

Exemplo incorreto colocando a operação no Path da API

O correto é que as rotas sejam apenas substantivos que representem o domínio em si e utilizem os verbos do protocolo HTTP, conforme o exemplo abaixo:

Roteamento

Requisição	Path	Objetivo
GET	/api/products/	Consultar todos produtos
GET	/api/people/	Consultar todas as pessoas
GET	/api/campaigns/	Consultar todas as campanhas

Exemplo de Path padronizado

Paths extensos

Evite Paths enormes, limite até no máximo 2 níveis de Paths aninhados ao principal, conforme o exemplo:

Roteamento		
Requisição	Path	Objetivo
GET	/api/brands/	Consulta todas as marcas
POST	/api/brands/	Cria uma Marca
GET	/api/brands/4/models	Consulta todos os modelos de uma marca
GET	/api/brands/4/models/30	Consulta um modelo especifico de uma marca
POST	/api/brands/4/models/30/versions	Cria uma versão para um modelo
PUT	/api/brands/4/models/30/versions/1	Autaliza uma versão

Exemplo de Path com variações em até 3 níveis

Padronização dos códigos de resposta

Se olharmos a tabela abaixo de códigos HTTP, vamos nos deparar com uma infinidade de códigos, implementa-los a risca se torna algo extremamente impossível, é um árduo trabalho tanto para quem constrói as Api's para quem as consome.

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request

207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
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
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

Tabela de códigos de retorno do protocolo HTTP

Para resolver este problema, geralmente cada desenvolvedor elege um grupo de códigos HTTP para utilizarem no desenvolvimento de suas API's, seguindo esta linha, recomendo o uso dos seguintes códigos:

HTTP 200 - OK

Geralmente esse é um código de retorno é utilizado como coringa, onde independente do tipo de requisição, se o processamento é realizado com sucesso, esse é o código de retorno default, porem eu recomendo utiliza-lo nos seguintes momentos:

1. Requisições do tipo GET onde o dado de retorno não esta paginado, por exemplo uma consulta por um id.
2. Requisições GET onde a API esta retornando a ultima pagina de um dado paginado.
3. Requisições POST que realizem operações lógicas que não criam dados, por exemplo um cálculo.

HTTP 201- Created

Geralmente utilizado em requisições POST, onde algum dado foi criado em algum repositório.

HTTP 202 - Accepted

Geralmente utilizado em requisições POST, indica que algum dado foi recebido e seu processamento será realizado de forma assíncrona , por exemplo a API posta o dado em uma fila MQ e um worker tratará o dado em um segundo momento.

HTTP 204 - No Content

Geralmente utilizado em requisições PUT ou DELETE, indica que o processamento foi finalizado e a API não tem nenhum dado para devolver para seu cliente.

HTTP 206 - Partial Content

Geralmente utilizado em requisições GET onde o dado retornado esta paginado

HTTP 400 - Bad Request

Geralmente utilizado em todos os tipos de requisições, ocorre quando o cliente preencheu algum dado incorreto na requisição.

Geralmente este é um código coringa, quando sua API não tratar individualmente outros códigos 4XX, a mesma deve retornar o código 400.

HTTP 401 - Unauthorized

Geralmente é utilizado para qualquer tipo de requisição, ocorre sempre que um usuário não esta autenticado para uso da API.

HTTP 404 - Not Found

Esse código de resposta pode ser utilizado nos seguintes momentos:

1. Pode ser utilizado para qualquer tipo de requisição, ocorre quando a aplicação cliente procura por uma rota/recurso que não existe
2. Pode ser utilizado em uma requisição GET ou PUT, ocorre quando a aplicação client tenta consultar ou alterar um domínio que não existe

HTTP 408 - Timeout

Geralmente é utilizado para qualquer tipo de requisição, ocorre sempre que uma API gera timeout em uma das suas operações.

HTTP 409 - Conflict

Geralmente é utilizado em requisições do tipo POST, ocorre quando o cliente tenta criar um dado que ja existe.

HTTP 500 - Internal Server Error

Geralmente é utilizado para qualquer tipo de requisição, ocorre para qualquer tipo de erro de processamento da API.

Geralmente este é um código coringa, quando sua API não tratar individualmente outros códigos 5XX, a mesma deve retornar o código 500.

HTTP 502 - Bad Gateway

Geralmente é utilizado para qualquer tipo de requisição, ocorrerá sempre que uma dependência externa a API, apresentar algum tipo de comportamento inesperado.

Lembrando que os códigos que descrevi acima são os conjuntos de códigos que geralmente utilizo, porem isso não é uma regra, se fizer sentido para sua API, utilize outros códigos.

Versionamento

Existem diferentes modelos de versionamento de API, não existe nenhuma regra definida no protocolo HTTP, também não existe nenhum tipo de padrão que seja unanimidade pela comunidade, seguindo esta linha, sugiro que sua utilização esteja bem explicita para os clientes, com a versão no final do Path, por exemplo:

Roteamento		
Requisição	Path	Objetivo
GET	/api/products/v1	Consultar todos produtos
GET	/api/products/v2	Consultar todos produtos
GET	/api/products/4/v1	Consultar produto específico
GET	/api/products/4/v2	Consultar produto específico
POST	/api/products/v1	Criar um produto
POST	/api/products/v2	Criar um produto

Exemplo de versionamento no Path

Uma pratica importante é garantir que o versionamento da API não fique diretamente em seu código fonte, deixe seu código fonte limpo, deixe essa responsabilidade com um Gateway de API, caso precise manter 2 diferentes versões da API no ar, tenha as diferentes versões publicadas em instancias diferentes da sua infraestrutura de produção.

Autenticação

Garantir a segurança dos dados também faz parte do desing de uma API, hoje o principal modelo utilizado pela comunidade é o Oauth 2.0, este é um padrão de autenticação, que foi construído para preservar os dados sensíveis de um usuário, minha recomendação seria sempre que possível utiliza-lo, porem caso não seja possível, não deixe sua API sem autenticação, no mínimo utilize uma autenticação do tipo basic, onde geralmente é criado um token em base64 a partir de um usuário e senha.

Stateless

Garanta que sua API não guarde nenhum tipo de estado referente a comunicação, toda requisição deve ser inteligente o suficiente para se resolver, sem armazenar quaisquer dados de seção.

Negociação de conteúdo

Apesar do ideal ser trafegar apenas JSON, caso sua API trafegue outros tipos de dados, por exemplo XML, utilize as chaves Accept e Content-Type do protocolo HTTP para detalhar todas as possibilidades

Paginação

Em requisições GET que retornem listas, devolva somente o necessário, pagine sua requisição, desta forma todos ganham, pois será trafegado somente o necessário, melhorando a experiencia do usuário final.

Não existe nenhum padrão amplamente utilizado pelo mercado para tal prática, por esse motivo sugiro que sua utilização esteja bem explicita para os clientes, por exemplo:

```
http://localhost:8080/api/customers?start=20&limit=5
```

No exemplo acima o campo ***start*** controla a partir de qual item deve se iniciar a consulta, ja o campo ***limit*** controla a quantidade máxima de registros na consulta.

Para uma paginação ser completa, não basta apenas paginar o request, precisamos também paginar de forma clara e explicita para os clientes o response, na pratica teríamos um retorno similar a este exemplo:

```
{
    "metadata": {
        "type": "list",
        "start": 20,
        "limit": 5,
        "total": 51
    },
    "results": [
        {
            "id": 21,
            "name": "Rishley Snyder",
            "sex": "M",
            "email": "rishley.snyder@hotmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1985-06-30",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 22,
            "name": "Milo Mccurdy",
            "sex": "M",
            "email": "milo.mccurdy@yahoo.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
```

```
            "date_birth": "1974-05-07",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 23,
            "name": "Sammi Zepeda",
            "sex": "F",
            "email": "sammi.zepeda@gmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1956-12-29",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 24,
            "name": "Romaine Dingman",
            "sex": "F",
            "email": "romaine.dingman@outlook.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1990-08-26",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 25,
            "name": "Almeda Hack",
            "sex": "F",
            "email": "almeda.hack@hotmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1981-09-05",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        }
    ]
}
```


No exemplo acima, na estrutura principal do JSON temos os campos ***results*** que contempla o conteúdo da requisição e o campo ***metadata***, detalhando os dados referente a paginação.

Ordenação

Otimize suas requisições, facilite ainda mais a vida de seus clientes, permita que sua Api, ordene sua resposta, desta forma o cliente pode não precisar realizar nenhum tipo de tratamento no dado.

Como no item anterior, não existe nenhum padrão amplamente utilizado pelo mercado para tal prática, por esse motivo sugiro que sua utilização esteja bem explicita para os clientes, por exemplo:

```
http://localhost:8080/api/customers?
start=0&limit=5&sort=sex,date_birth&desc=date_birth
```

No exemplo acima o campo ***sort*** controla quais os campos devem ser ordenados separados por virgula, ja o campo ***desc*** controla quais os campos devem ser ordenados de forma decrescente, desta forma podemos considerar que a ordenação default é crescente.

Na solicitação ordenamos pelo campo ***sex*** e ***date_birth***, ainda considerando o campo ***date_birth*** na forma decrescente, na pratica teríamos um retorno similar a este exemplo:

```
{
    "metadata": {
        "type": "list",
        "start": 0,
        "limit": 5,
        "total": 51
    },
    "results": [
        {
            "id": 49,
            "name": "Janna Ying",
            "sex": "F",
            "email": "janna.ying@yahoo.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1992-06-06",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 24,
            "name": "Romaine Dingman",
            "sex": "F",
            "email": "romaine.dingman@outlook.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1990-08-26",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
```

```
        },
        {
            "id": 7,
            "name": "Ruthie Coco",
            "sex": "F",
            "email": "ruthie.coco@gmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1988-07-27",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 37,
            "name": "Scarlet Barnwell",
            "sex": "F",
            "email": "scarlet.barnwell@hotmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1987-10-26",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        },
        {
            "id": 9,
            "name": "Vinaya Justus",
            "sex": "F",
            "email": "vinaya.justus@hotmail.com",
            "cel_ddi": "55",
            "cel_ddd": "11",
            "cel_number": "987654321",
            "date_birth": "1986-04-18",
            "register_date": "2018-09-06 22:32:51",
            "last_update_date": null
        }
    ]
}
```


Respostas Parciais

Otimize ainda mais suas requisições GET, permita seus clientes recuperarem somente os campos que realmente necessitam, desta forma de ponta a ponta serão trafegado apenas os dados necessários.

Como nos itens anteriores, não existe um padrão amplamente utilizado pela comunidade, por isso eu sugiro que seja algo bem explicito, para que facilmente os clientes de sua API consigam identificar, por exemplo:

```
http://localhost:8080/api/customers/1?
fields=metadata[],results[][id,name,sex]
```

No exemplo acima o campo **fields** controla quais os campos devem ser retornados no retorno da requisição, separando os campos no nível principal por virgula e os campos aninhados através do [], na pratica teríamos um retorno similar a este exemplo:

```
{
    "metadata": {
        "type": "object",
        "start": 0,
        "limit": 0,
        "total": 0
    },
    "results": {
        "id": 1,
        "name": "Willian da Silva",
        "sex": "M"
    }
}
```

Hateoas

Hateoas representa o termo Hypermedia as the engine of application State (Hipermídia como o mecanismo do estado do aplicativo), é um padrão que sempre inclui links uteis junto as respostas de uma requisição, por exemplo:

```
http://localhost:8080/api/customers?start=20&limit=2
```

Como nos itens anteriores, não existe um padrão amplamente utilizado pela comunidade, o único consenso, é que a informação fica centralizada no objeto **links**, por exemplo:

```
{
   "metadata":{
      "type":"list",
      "start":20,
      "limit":2,
      "total":51
```

```
    },
    "results":[
       {
          "id":21,
          "name":"Rishley Snyder",
          "sex":"M",
          "email":"rishley.snyder@hotmail.com",
          "cel_ddi":"55",
          "cel_ddd":"11",
          "cel_number":"987654321",
          "date_birth":"1985-06-30",
          "register_date":"2018-09-06 22:32:51",
          "last_update_date":null,
          "links":{
             "self":{

"href":"http://localhost:8080/api/customers/21",
                "type":"GET"
             },
             "delete":{

"href":"http://localhost:8080/api/customers/21",
                "type":"DELETE"
             }
          }
       },
       {
          "id":22,
          "name":"Milo Mccurdy",
          "sex":"M",
          "email":"milo.mccurdy@yahoo.com",
          "cel_ddi":"55",
          "cel_ddd":"11",
          "cel_number":"987654321",
          "date_birth":"1974-05-07",
          "register_date":"2018-09-06 22:32:51",
          "last_update_date":null,
          "links":{
             "self":{

"href":"http://localhost:8080/api/customers/22",
                "type":"GET"
             },
             "delete":{

"href":"http://localhost:8080/api/customers/22",
                "type":"DELETE"
             }
          }
       }
    ]
}
```


Os principais benefícios de seguir essa abordagem são:

1. As aplicações clients conhecem facilmente o limite e próximos passos de cada domínio

2. As aplicações clients podem recuperar seus paths dinamicamente sem a necessidade de colocar-los fixo no código

3. Os desenvolvedores da API podem alterar o esquema de paths sem quebrar as aplicações clients

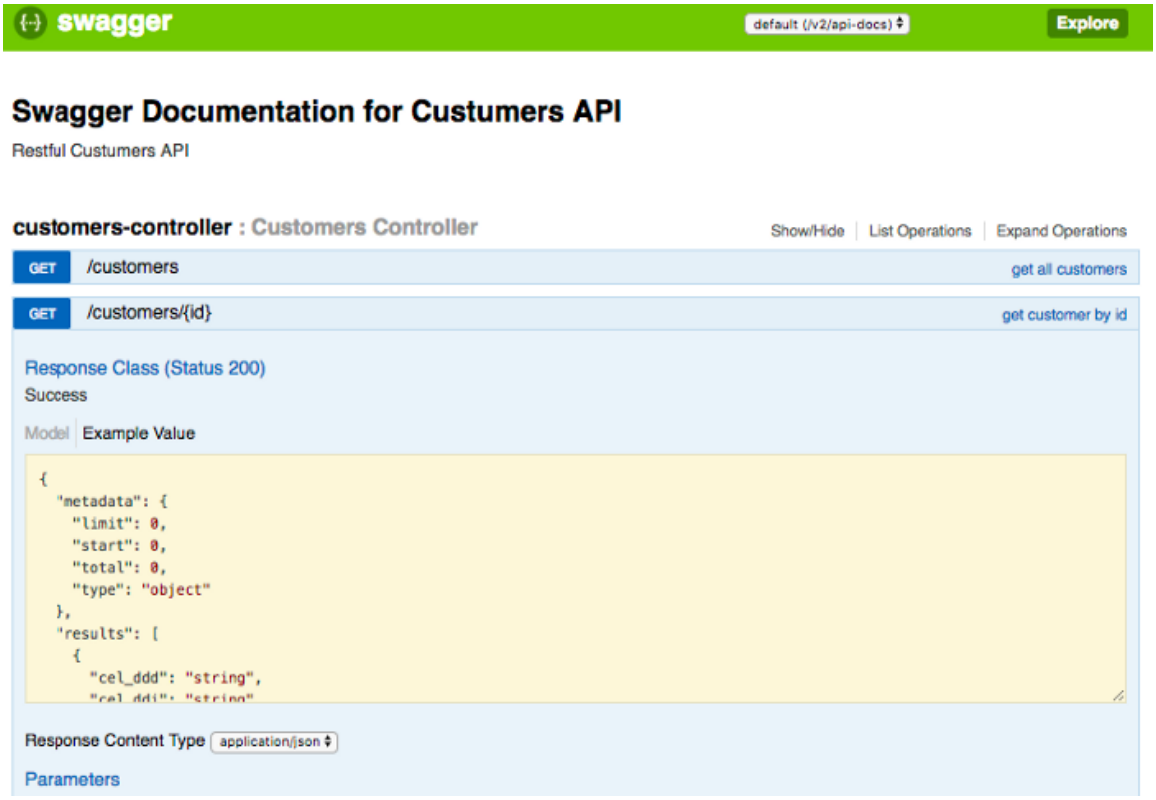
Open Documentation

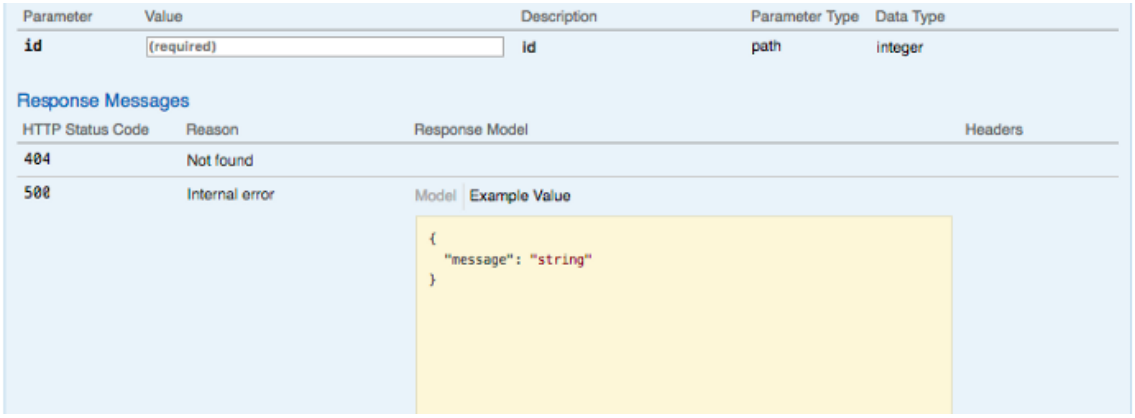
Alem de seguir as melhores práticas, faça ainda mais que isso, utilize frameworks como o Swagger, que é um framework open source que apoia o desenvolvedor no desenho, documentação e especificação de suas Api's.

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.

Uma outra grande vantagem do Swagger é que a partir de sua especificação, é possível gerar clients para consumo das Api's, diminuindo a complexidade das integrações e o esforço para desenvolvimento.

Veja abaixo um exemplo de uma API com Swagger:





Exemplo documentação com Swagger

Conclusão

Se você leu atentamente todos os itens citados acima, você ja tem a total certeza que para construir uma API Rest de alta qualidade, não basta apenas conhecer uma linguagem de desenvolvimento, é preciso conhecer muito bem o protocolo HTTP e um conjunto de padrões que com certeza elevam o nível de sua API.

> ***Referencias bibliográficas***
>
> *Projetando uma API Rest — https://blog.octo.com/pt-br/projetando-uma-api-rest/*
>
> *Guia para projeto API HTTP — https://github.com/Gutem/http-api-design/*
>
> *Rest Api Desing — Resource Modeling — https://www.thoughtworks.com/pt/insights/blog/rest-api-design-resource-modeling*
>
> *Richardson Maturity Model — https://martinfowler.com/articles/richardsonMaturityModel.html*
>
> *RFCs do Protocolo HTTP — https://tools.ietf.org/html/rfc2616*
> *https://tools.ietf.org/html/rfc7540*
> *https://tools.ietf.org/html/rfc7541*
>
> *Introdução ao HTTP/2 — https://developers.google.com/web/fundamentals/performance/http2/*
>
> *HTTP — https://developer.mozilla.org/pt-BR/docs/Web/HTTP*