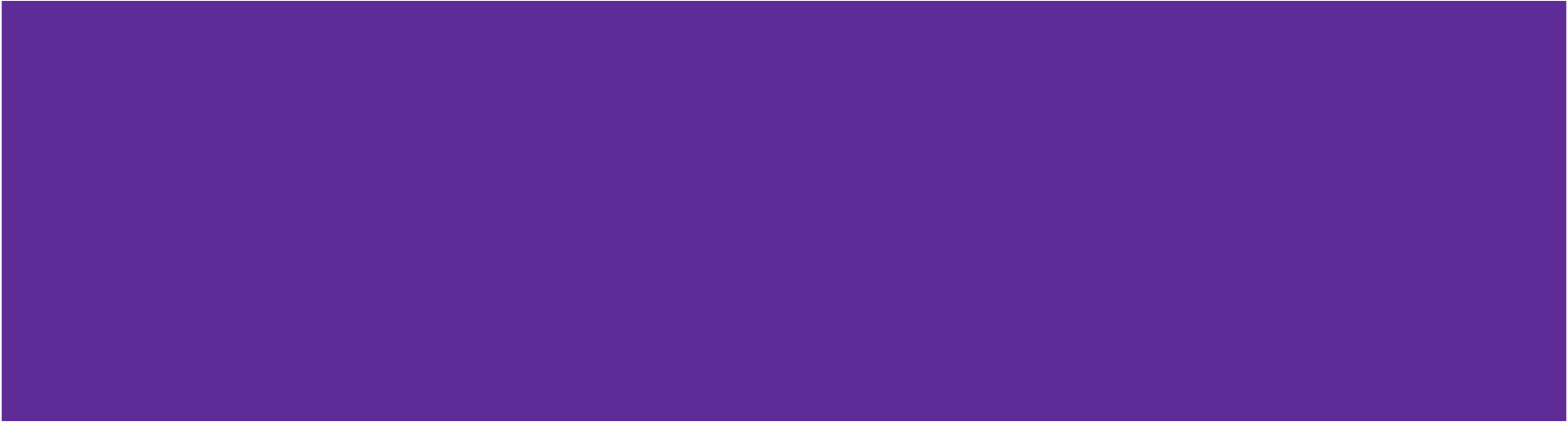


Design de API's

Como deixar desenvolvedores felizes ao utilizarem sua API?
TDC Connections 2022



Quem sou eu?

Desenvolvedor com 5 anos de experiência.

Recifense.

Integrante do time de Ecossistemas e API da Nuvemshop(temos vagas!)

Sobre o que vamos e o que não vamos falar?

Vamos falar de...

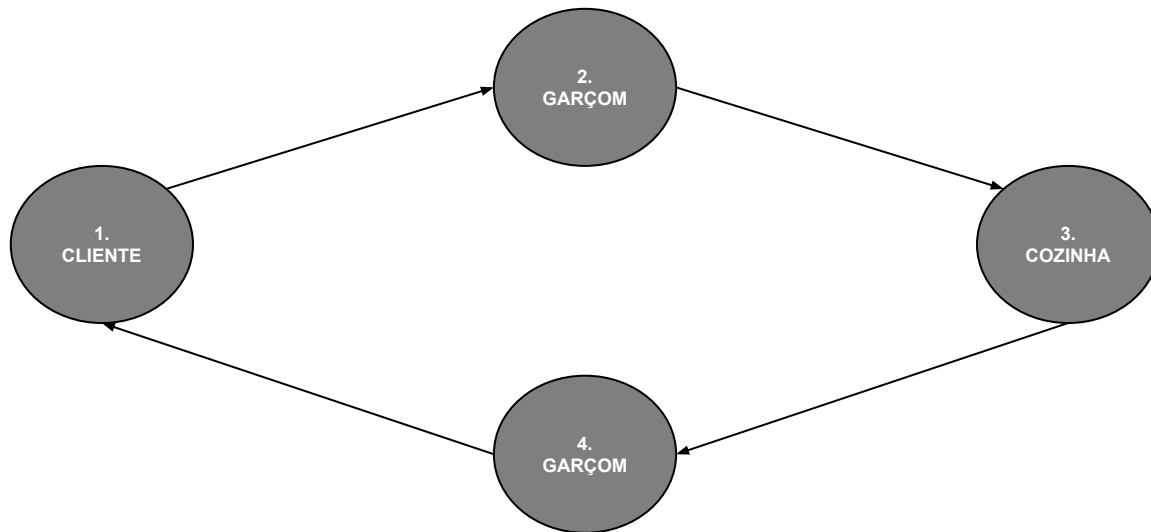
- Quando minha API é Restful?
- A importância de recursos bem definidos para os clientes
- Boas práticas na implementação destes recursos(*endpoints*)
- Entendendo idempotência e método HTTP seguro
- *Status Code* podem nos dizer muito!
- Erros comuns e evitáveis
- Alguns pontos sobre DX para API's

Não vamos falar sobre...

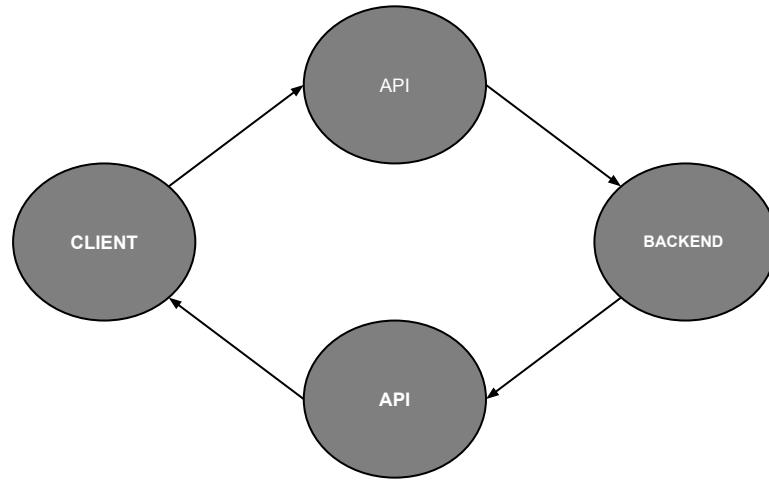
- Segurança de API's
- Tipos e uso de *headers*(*Content negotiation, compression, etc*)
- *caching*
- Versionamento

O que é uma API?

- Application Programming Interface
 - interface de programação de aplicativos



Na vida de desenvolvimento....



Quando uma API é restful?

- HTTP/HTTPS
- **Um serviço *Restful* implementa o padrão REST**
 - Arquitetura cliente-servidor
 - Comunicação *stateless*
 - *Cache*
 - Arquitetura em camadas
 - *HATEOAS*

```
{  
  "id": "7a8373-2837-45f8-a1d7-8fea0bed5d38",  
  "name": "Hatchi",  
  "age": 8,  
  "breed": "akita",  
  "_links": [{  
    "rel": "self",  
    "href": "https://myapi.com/v1/dogs/7a8373-2837-45f8-a1d7-8fea0bed5d38",  
    "method": "PUT"  
  }]  
}
```

Não implementei HATEOAS... minha API não é *restful*?

- O modelo de maturidade de *Richardson* para a chegar à glória do REST

SWAMP OF POX	RESOURCES	HTTP VERBS	GLORY OF REST 🙌
A aplicação só tem um <i>endpoint</i> . E quem define as ações são as requisições.	Os recursos já são separados em <i>endpoints</i> , mas a utilização de verbos HTTP ainda não segue um padrão.	Nível onde a maioria das APIS se encontram. Verbos HTTP são usados de forma semântica.	Padrão REST implementado com sucesso, segundo <i>Roy Fielding</i> .

Recursos

- São acessados através de URLs HTTP.
- Contém um tipo.
- Contém dados.
- Podem conter outros recursos como forma de relacionamento.

Verbos HTTP*

- **POST**

- Criar um recurso

- **PUT**

- Criar ou atualizar um recurso

- **DELETE**

- Deletar um recurso



- **GET**



- Buscar um recurso

- **PATCH**

- Atualizar parcialmente um recurso

Boas práticas



- HTTP /api/:version/:recurso/:identificador 
- HTTP /api/:recurso/:identificador/:subrecurso/:identificador 

- HTTP /api/:recurso/:identificador/atualizar 
- HTTP /api/:recurso/:identificador/:identificador 




Provendo recursos de sua API de forma semântica

"Preciso fornecer em minha API, uma lista de cães e também fornecer apenas um cão através de um identificador único"

Sugestão...

- **GET** /dogs  HTTP 200
- **GET** /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02  HTTP 200

Seria bom evitar...

- /dogs/list 
- /getdogs/ 
- /dog/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 

GET - /dogs

Se tiver recursos:

- A lista de recursos
- Status 200 OK

Se não tiver recursos:

- A lista vazia
- Status 200 OK

```
[
  {
    "id": "4h5693-3145-183f0-a2v0-37ea1bvv9f3c",
    "name": "Scooby Doo",
    "age": 5,
    "breed": "great dane"
  },
  {
    "id": "7a8373-2837-45f8-a1d7-8fea0bed5d38",
    "name": "Hatchi",
    "age": 8,
    "breed": "akita"
  }
]
```

GET - /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02

Se tiver recurso:

- Retornar o recurso no response body
- Status 200 OK

Se não tiver o recurso:

- Status 404 NOT FOUND

```
{  
  "id": "7a8373-2837-45f8-a1d7-8fea0bed5d38",  
  "name": "Hatchi",  
  "age": 8,  
  "breed": "akita"  
}
```

Salvando um novo recurso na API

"Preciso fornecer na minha API uma forma de salvar um novo dado de cachorro, e também atualizar ele quando necessário"

- **PUT** /dogs HTTP **201** 🤔

```
{  
  "name": "Scooby Doo",  
  "age": 4,  
  "breed": "Great Dane"  
}
```

- **PUT** /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5do2 HTTP **200** 🤔

```
{  
  "name": "Scooby Doo",  
  "age": 5,  
  "breed": "Great Dane"  
}
```

Desvantagens para a API...

- Um único método para duas responsabilidades
- Se o objeto fosse deletado anteriormente, ao requisitar uma atualização, a API seria forçada a criar um novo recurso, utilizando o id gerado pelo *client*.
- Poderíamos ter inconsistências na **idempotência**.

Idempotência e segurança dos verbos

- Uma operação idempotente é aquela em que o resultado sempre será o mesmo não importa quantas vezes ela for chamada.
- Em API's, segundo o protocolo **HTTP**, a idempotência está relacionada com o efeito na aplicação da intenção que a requisição tem.

Idempotência X segurança dos verbos

- Idempotência significa que o resultado de um requisição bem sucedida não altera a intenção da mesma independente da quantidade de vezes que ela foi chamada.
- Um método seguro **não altera um recurso**.

Método	Idempotente	Seguro
POST	✗	✗
PUT	✓	✗
GET	✓	✓
DELETE	✓	✗
PATCH	✗	✗

POST - /dogs

Em caso de sucesso:

- Status 201 CREATED
- id do recurso criado no **response body**

Em caso de falha por request inválido do *client*:

- Status 400 BAD REQUEST ou 422 UNPROCESSABLE ENTITY
- Lista de erros a serem ajustados no request

Em caso de falha por erro do servidor:

- Status 500 INTERNAL SERVER ERROR

```
{  
  "name": "Scooby Doo",  
  "age": 4,  
  "breed": "Great Dane"  
}
```

PUT - /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02

Em caso de sucesso:

- Status 200 OK ou 204 NO CONTENT
- id do recurso alterado no **response body**

Em caso de falha por request inválido do *client*:

- Status 400 BAD REQUEST ou 422 UNPROCESSABLE ENTITY ou 409 CONFLICT EXCEPTION
- Lista de erros a serem ajustados no request

Em caso de falha por erro do servidor:

- Status 500 INTERNAL SERVER ERROR


```
{  
  "name": "Scooby Doo",  
  "age": 5,  
  "breed": "Great Dane"  
}
```

Evite essas sintaxes

- PUT /dogs/update/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 ✗
- POST /dogs/create ✗
- PUT /dog/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 ✗
- POST /dog ✗

Não quero ter que enviar todo o objeto para atualizar...

Com o **PATCH**, todos os campos podem ser opcionais, mas pelo menos um deve ser enviado na requisição.

- **HTTP PATCH** /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 

Em caso de sucesso:




- HTTP 200

Em caso de erro:

- HTTP 400
- HTTP 500
- HTTP 404



O delete não tem segredo

- DELETE /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 
- DELETE /dogs/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02/remove 
- DELETE /dog/02a34ab3-9a03-45f8-a1d7-4fee0bbd5d02 

Em caso de sucesso:

- HTTP 200 ou HTTP 204

Ajudando o *client* a buscar o que ele precisa

- *Query parameters*

```
/dogs?breed=doberman  
/dogs?breed=doberman&age=3
```

Em caso de erro:

- HTTP 400 Bad Request

Ajudando o *client* a buscar o que ele precisa

- *Field Projection*

```
/dogs?fields=name,breed  
/dogs/0sk19a0s-6fhkjms-72h18cn-827zkmdsa?fields=name,breed
```

EM CASO DE ERRO, DUAS ALTERNATIVAS:

- Ignorar o campo inválido e trazer apenas os válidos
- Lançar HTTP 400 BAD REQUEST

EM QUALQUER UM DESTES CASOS, A ALTERNATIVA DEVE ESTAR DOCUMENTADA.

Ajudando o *client* a buscar o que ele precisa

- *Sorting & ordering*

```
/dogs?sort=createdAt&order_by=desc
```

- *Pagination*

```
/dogs?take=10&skip=2
```

Verbos HTTP pra turbinar sua API

- Atualizar um recurso com o estado já alterado no servidor - **HTTP 409 CONFLICT**
- Um formulário correto sintaticamente porém com erros em regra de negócio - **HTTP 422 UNPROCESSABLE ENTITY**
- Criar ou atualizar um recurso de forma assíncrona - **HTTP 202 ACCEPTED**
- Um *client* tentou acessar um recurso que existe mas é de outro *client* - **HTTP 404 NOT FOUND**

Erros comuns e evitáveis

- Implementar a API apenas como um *server* para um *frontend*
- Não criar contratos para os objetos trafegáveis de requisição e resposta
- **Não documentar sua API**
 - OpenApi
 - *Swagger*

Developer Experience em sua API

- **Chamadas por objetivo** - Quantidade de chamadas para atingir um objetivo?
- **Estrutura** - A estrutura de suas requisições e respostas são claras?
- **Navegação** - Quão complexo é navegar entre os recursos?
- **Dependências** - Sua API tem muitas dependências pra rodar?
- **Tempo do primeiro *request***
- ***Error handling***

Bora conversar! Dúvidas ou sugestões?

LinkedIn: Adeildo Neto

Github: Nethanos

MUITO OBRIGADO!