Licenciatura
Multimédia

Imagem retirada do website
https://unsplash.com/photos/MuJHwDHbXUk

Programação III

Frederico Fonseca
fffonseca@ismt.pt

INSTITUTO SUPERIOR
MIGUEL TORGA

Sumário

- *Application Programming Interface* (API)
- Web API
- REST e RESTful
- OpenAPI
- Boas práticas

API

- API = *Application Programming Interface* (API);
- Uma API (Application Programming Interface) é um conjunto de regras e definições que permitem que diferentes sistemas ou aplicações comuniquem entre si, geralmente através de *endpoints* acessíveis por meio de requisições programáticas;
- As API privilegiam a **interface máquina-máquina** e permitem aos programadores aceder a funcionalidades de outras aplicações/serviços, através de uma estrutura de dados bem definida;

Figura 1 – Exemplificação do funcionamento de uma API

API

- Os autores das aplicações ou serviços criam, por norma, uma API específica para os outros programadores;
  - Permitindo criar plugins, estendendo-lhes, assim, as funcionalidades do programa;
- Exemplos de APIs:
  - **PayPal** - https://developer.paypal.com/docs/api/overview/
  - **Twitter** - https://developer.twitter.com/en/docs/twitter-api/tweets/lookup/
  - **OpenWeather** - https://openweathermap.org/api
  - **Skyscanner** - https://skyscanner.github.io/slate/#api-documentation

The Movie DB

- O '**The Movie DB**' (TMDB) é uma API que contém toda a informação sobre filmes e séries;
- Iremos utilizar esta API nas nossas aulas;

https://www.themoviedb.org

Web API

- Uma API pode ser desenvolvida de diversas maneiras e formatos;
- Utilizam-se **padrões** consoante o objetivo da aplicação, por exemplo:
  - World Wide Web (WWW)
    - O padrão maioritariamente adotado é o **REST** - *Representational State Transfer*;
      - Inclui um conjunto de métodos pré-definidos: GET, POST, PUT, DELETE, HEAD e PATCH;
- Outros padrões: **SOAP** *(Simple Object Access Protocol)*, ***FIX Protocol*** *(na área financeira)*, *GraphQL*, *entre outros*;

Web API

- É uma API aplicada ao contexto **Web**;
- Ao desenvolver uma aplicação o programador pode ter acesso, via API, a diversas outras aplicações/serviços;
- Conjunto definido de mensagens de pedido e resposta HTTP;
  - Podem ser expressos no formato XML ou **JSON** (preferencial);
  - Atualmente os serviços Web são maioritariamente desenvolvidos utilizando a arquitetura ***Representational State Transfer*** (REST);

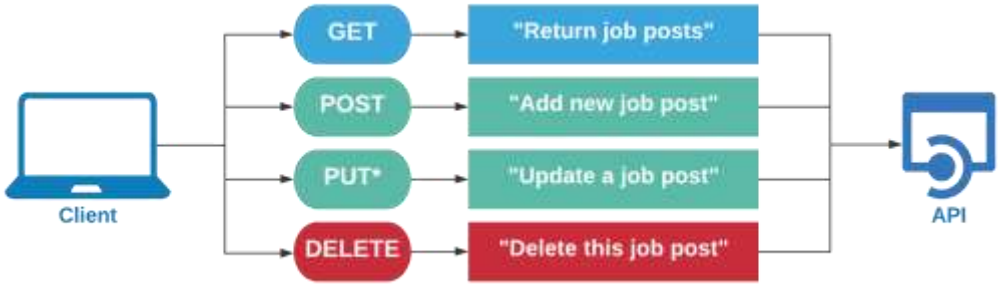


Figura 2 – Exemplificação dos possíveis estados de um pedido REST.

Web API

- Vantagens da **arquitetura REST** numa **Web API**:
  - Eficiência;
  - Diversidade de aplicações;
  - Gestão de processos;
  - Automatização de procedimentos;
  - Interoperabilidade e integração;
  - Personalização;
  - Entre outros.

RESTful API

GET PUT POST DELETE

REST

- REST (*Representational State Transfer*) é uma arquitetura que define princípios e restrições para a construção de serviços Web escaláveis e interoperáveis;
- Os sistemas compatíveis com REST são denominados de **RESTful**;
- Caracterizam-se pelo facto de:
  - não terem estado (*stateless*);
  - separarem as camadas do cliente e do servidor;

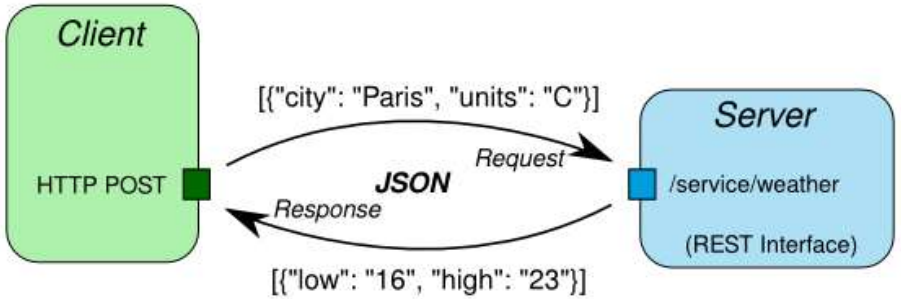


Figura 3 – Descrição do processo de execução de um pedido a uma API.

REST vs RESTful

- REST é uma arquitetura para desenvolvimento de APIs;
  - Define os princípios que estruturam a comunicação entre sistemas distribuídos;
- RESTful é a implementação correta dos conceitos REST numa API;
  - Nem toda a API que usa pedidos HTTP é RESTful;

REST

- Esta arquitetura pressupõe que as implementações do cliente e do servidor sejam efetuadas de forma independente;
  - Sem que um tenha conhecimento do outro;
- O código do lado do cliente pode ser alterado a qualquer momento sem que isso afete a operação do lado do servidor, e vice-versa;
- Para que a comunicação seja realizada com sucesso, apenas é necessário saber o **formato das mensagens** usado para cada um dos lados;

REST

- Exemplo de um endpoint da API REST de Projetos do **GitHub**:
  - https://docs.github.com/en/rest

List organization projects

Lists the projects in an organization. Returns a `404 Not Found` status if projects are disabled in the organization. If you do not have sufficient privileges to perform this action, a `401 Unauthorized` or `410 Gone` status is returned.

```
GET /orgs/:org/projects
```

Name	Type	Description
`state`	`string`	Indicates the state of the projects to return. Can be either `open`, `closed`, or `all`. Default: `open`

Response

REST

- Uma pedido REST necessita de:
  - Um **método HTTP** para definir a operação a ser executada;
  - Um **cabeçalho**, onde o cliente passa as informações sobre o pedido;
  - Um **URL** (*endpoint*) com o caminho para um recurso REST;

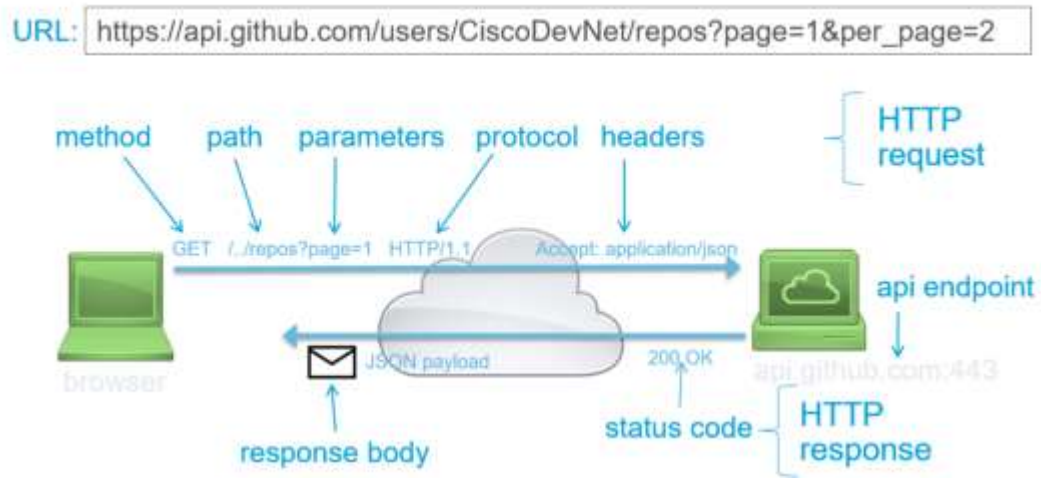
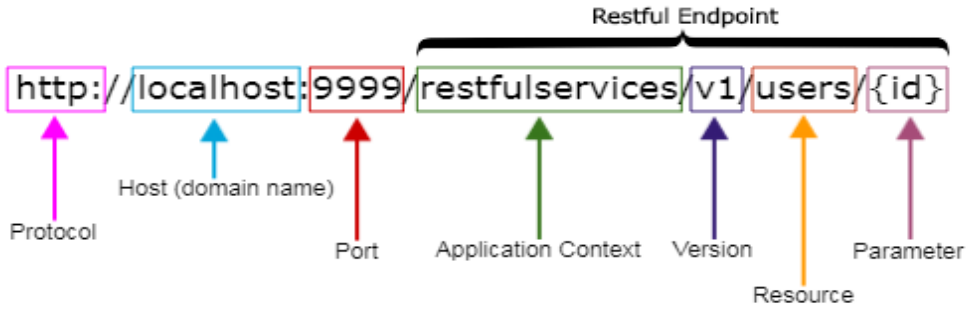


Figura 4 – Exemplificação de um pedido HTTP em REST.

REST

- Existem **quatro métodos HTTP** para interagir com coleções/recursos num sistema REST:
  - **GET** - lê um dado específico (por ID) ou um conjunto de dados;
  - **POST** - cria um novo dado;
  - **PUT** - atualiza um dado específico (por ID);
  - **PATCH** – para atualizações parciais;
  - **DELETE** - remove um dado específico (por ID);



REST

- As **coleções** (*collections*) são um conjunto de recursos;
- O URI é o caminho (*path*) através do qual um recurso pode ser localizado e onde nesse recurso as ações podem ser efetuadas;
  - Todos os recursos são acedidos através do URI;

Figura 5 - Exemplificação de um pedido HTTP a um endpoint específico. A resposta HTTP é uma coleção no formato JSON.

REST

- Os URL de pedidos não devem conter verbos ou nome de métodos;
  - Por exemplo, um *endpoint* '/adicionaNovoAluno' não é correto, o correto seria '/aluno';

PEDIDO	**Método HTTP**	**URL DO PEDIDO**	**Extra**	**RESPOSTA** *(http code)*
create	POST	http://localhost/api/aluno	{body}	201
read (todos os dados)	GET	http://localhost/api/alunos		200
read (de um ID específico)	GET	http://localhost/api/aluno/**:id**		200
update	PUT	http://localhost/api/aluno/**:id**	{body}	200
delete	DELETE	http://localhost/api/aluno/**:id**		204

REST

- Existem duas formas de desenvolver API's:
  - Contract-First - é exposto um contrato de serviço apenas com a definição da interface para programadores;
    - É a solução mais recomendada, porque documenta-se a especificação da API antes de efetivamente implementá-la;
  - Contract-Last - implementar todo o serviço e só depois nos preocupamos em documentar;
    - É a pior solução de todas, mas infelizmente a mais utilizada, com todos os problemas que daí advém;

REST

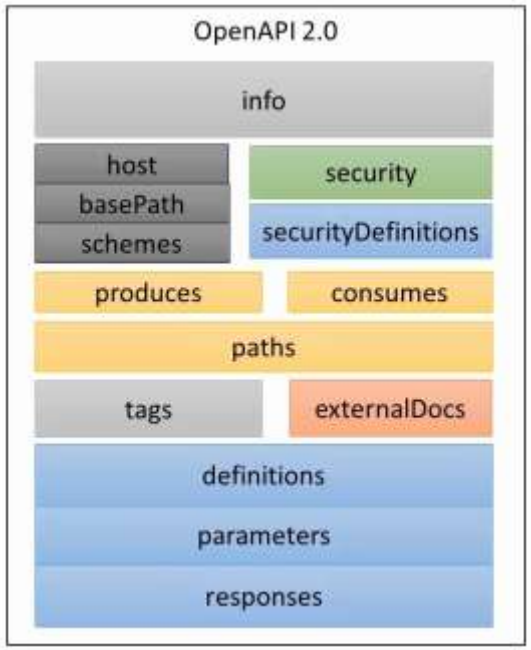
- Esta solução inclui a definição/exemplificação/geração de toda a estrutura necessária para a API;
  - I.e., a descrição de como o serviço é disponibilizado e como pode ser consumido de forma correta;
- A descrição aborda numa fase inicial, os detalhes dos recursos, como:
  - Endpoint (URI);
  - Operações disponíveis e como trabalhar com as funções;
  - Modelos de dados relacionados;
  - *Content-type* do *request* e *response*;
  - Entre outros detalhes relevantes;

Open API

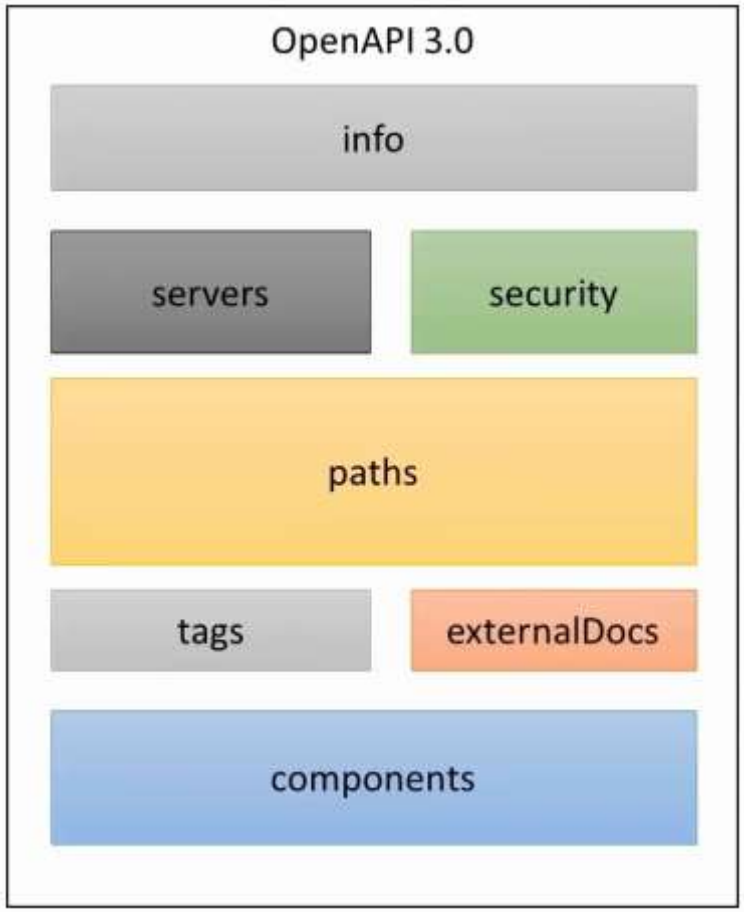
- Open API *(ou API Aberta)* - https://www.openapis.org
  - Suportado pela **Linux Foundation**, com a chancela da *Open API Initiative*;
    - Criado em 2010 pela Wordnik foi originalmente conhecido como ***Swagger Specification***;
    - Em 2015 foi comprado pela *SmartBear Software* e doado à *Linux Foundation*;

Open API

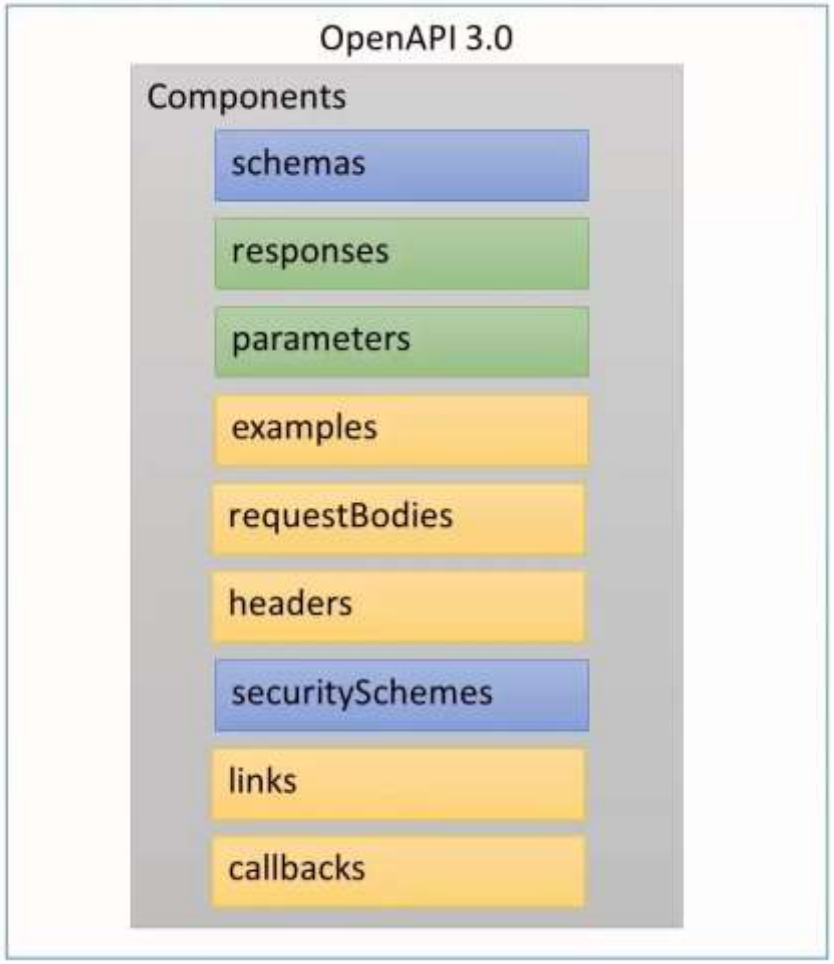
(old)



(new)



Open API



Open API

- Define uma **interface padrão**, independentemente da linguagem de programação, para APIs REST que disponibiliza tanto para sistemas como para programadores um mecanismo para mais facilmente compreenderem os serviços disponíveis na API sem acesso direto ao código-fonte;
- O conceito do "Open" tem a ver com o facto da API estar disponível para outras empresas e programadores;
  - Uma determinada organização cria uma API aberta e gratuita *(embora possa ser paga)*;
  - Como é o caso do Facebook, Twitter, SIBS API Market (PT), etc.;

Recursos

- **Swagger Editor**

  https://editor.swagger.io

- Documentação do Swagger Editor

  https://swagger.io/docs/specification/basic-structure/

- Tutorial sobre o OpenAPI (EN)

  https://idratherbewriting.com/learnapidoc/restapispecifications.html

- Tutorial do Swagger Editor (EN)

  https://idratherbewriting.com/learnapidoc/pubapis_swagger.html

BEST PRACTICES

Boas Práticas

- O design de uma API segue um modelo chamado ***Resource Oriented Design*** (ou design orientado a recursos);
- Este modelo consiste em 3 atributos chave:
  - **Recurso** (resource): um recurso é um dado;
    - Por exemplo, um **aluno**;
  - **Coleção** (collection): um grupo de recursos é denominado de coleção;
    - Por exemplo, uma **lista de alunos**;
  - **URL** (endpoint): identifica a localização do recurso ou coleção;
    - Por exemplo, '**/aluno**';

Boas Práticas

- O nome das URLs segue o conceito de "kebab-case";
  - Por exemplo, uma **lista de pedidos de um aluno** deve ter a URL '/pedidos-aluno' (ou em inglês, '/student-orders') e não '/systemOrders' ou '/system_orders' (está errado!);
- Os parâmetros devem utilizar o conceito de *camelCase*, isto é, "orderId" e não "OrderId";
  - Numa URL poderia ficar '/pedidos-aluno/:idPedido' (ou '/student-orders/:orderId');
  - Este conceito também é utilizado nas propriedades JSON.

Boas Práticas

- Para obtermos todos os alunos, a URL deve chamar-se '/alunos' e não '/aluno';
  - Recomenda-se uma propriedade chamada "total" que indique o número total de registos retornados. Por exemplo,

```
{
  "status": 200,
  "message": "sucess",
  "data": {
    alunos: [...],
    total: 34
  }
}
```

- Uma URL inicia com uma **coleção** e termina com um **parâmetro**;
  - Por exemplo, '/aluno/:idAluno';
  - Recomenda-se que a API aceite os parâmetros "offset" e "limit";
  - Por exemplo, '/alunos?offset=5&limit=5' (muito útil para paginação);

Boas Práticas

- URLs funcionais (*de monitorização*)
  - Os serviços RESTful podem implementar *endpoints* como '/health', '/version' ou '/metrics';
  - Estes fornecem a seguinte informação:
    - '/health' - responde com um código de status 200 OK;
    - '/version' - responde com o número da versão atual da API;
    - '/metrics' - fornece várias métricas, como o tempo médio de resposta, entre outros;
  - '/debug' e '/status' são endpoints também recomendados;

Boas Práticas

- Devemos usar recursos encadeados na URL.
- Alguns **exemplos práticos** são:
  - '/aluno/2/disciplinas' (*GET*) - obtenha a lista de todas as disciplinas frequentadas pelo aluno 2;
  - '/aluno/2/disciplina/31' (*GET*) - obtenha os detalhes da disciplina 31, que pertence ao aluno 2;
  - '/aluno/2/disciplina/31' (*DELETE*) - deve eliminar a disciplina 31, que pertence ao aluno 2;
  - '/aluno/2/disciplina/31' (*PUT*) - deve atualizar as informações da disciplina 31;