UPskill

Digital Skills & Jobs

Introduction to Web Development

Programação Orientada a Objetos

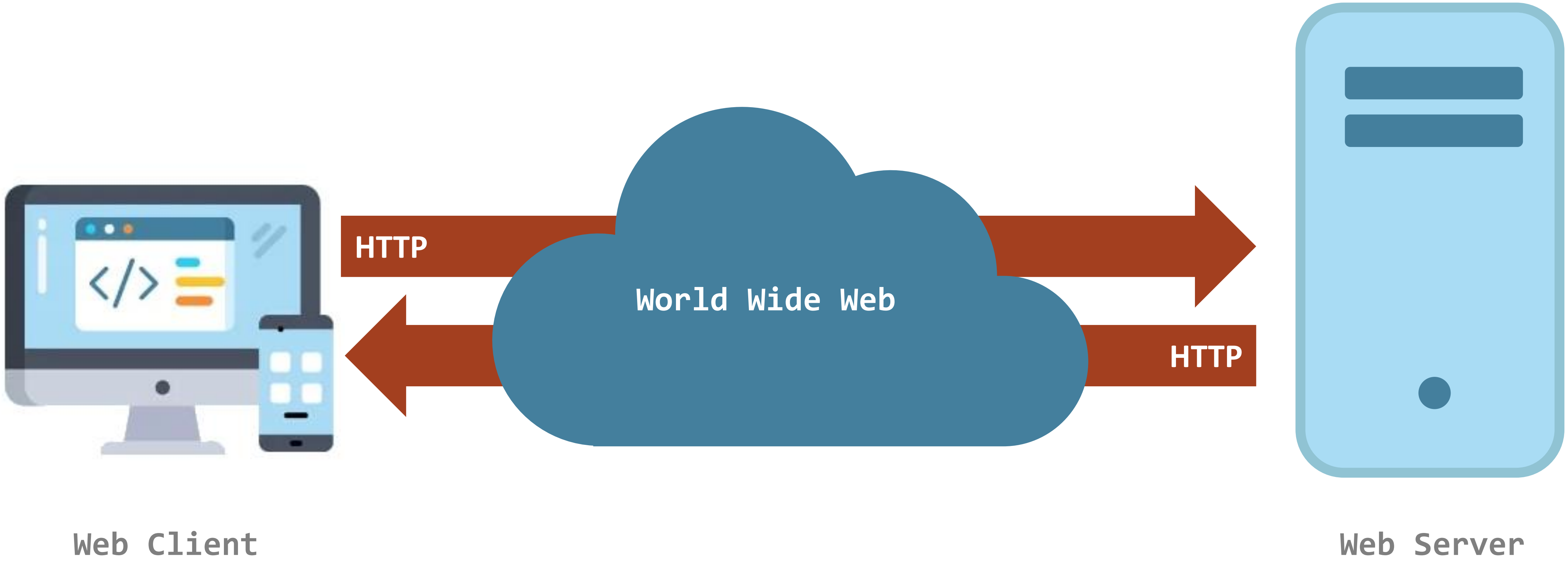
P.PORTO

apdc
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

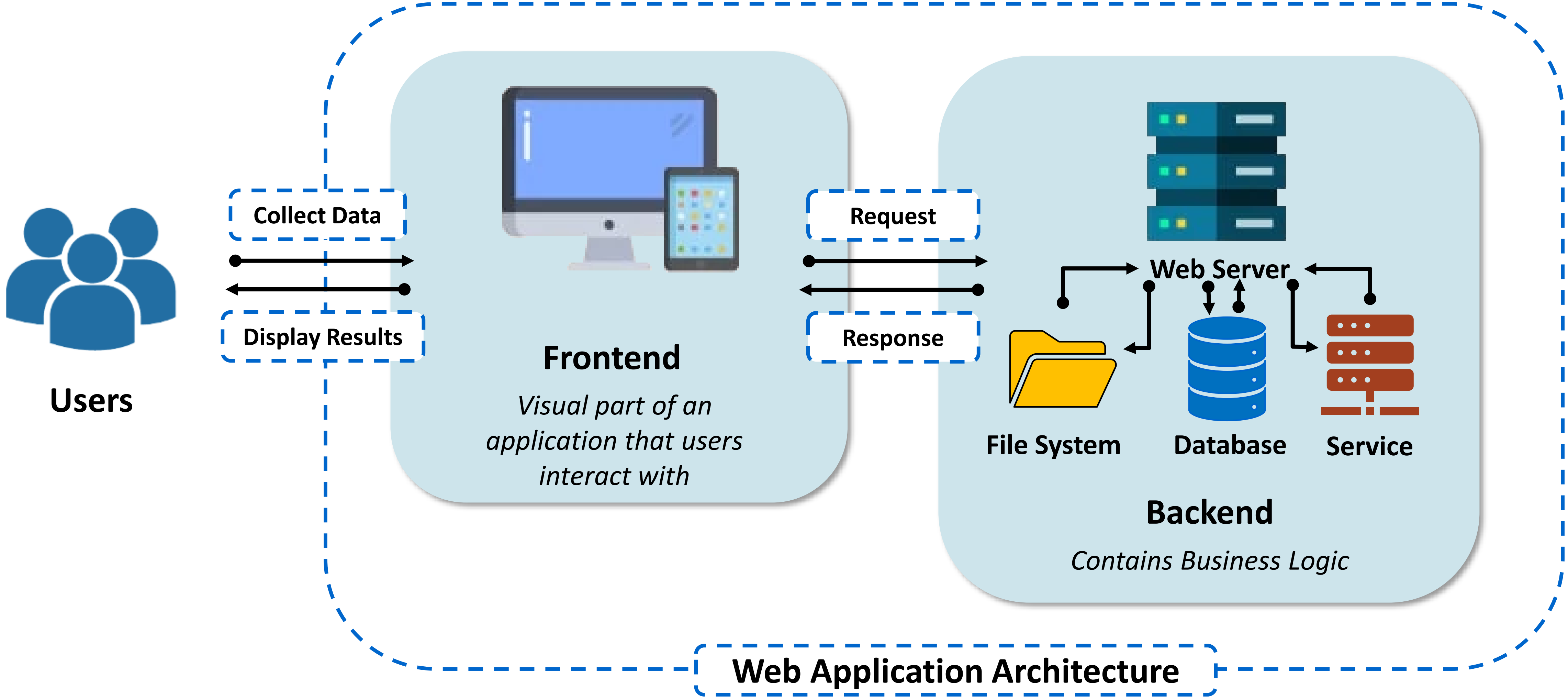
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

Web Architecture



Web Application Architecture



Hypertext Transfer Protocol

- The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems
- HTTP is the foundation of data communication for the World Wide Web
- HTTP development is a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C)
  - 1997 – HTTP/1 (version 1.1)
  - 2015 – HTTP/2
  - 2022 – HTTP/3 (Proposed Standard – RFC:9114 – June 2022)

Hypertext Transfer Protocol

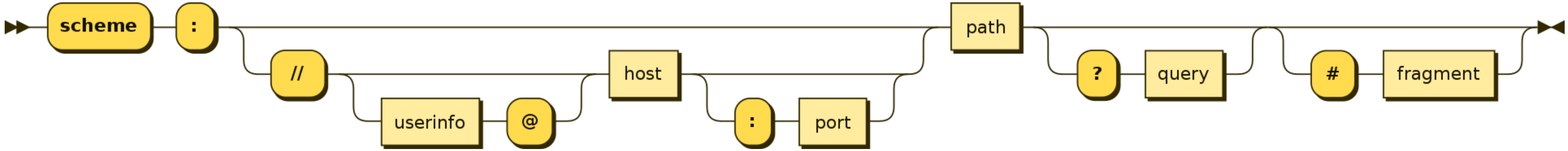
- Request methods
  - **GET** - Is used to retrieve information from the given server using a given URI.
  - **HEAD** - Same as GET but transfers the status line and header section only.
  - **POST** - Is used to send data to the server.
  - **PUT** - Replaces all current representations of the target resource with the uploaded content.
  - **DELETE** - Removes all current representations of the target resource given by a URI.
  - **CONNECT** - Establishes a tunnel to the server identified by a given URI.
  - **OPTIONS** - Describes the communication options for the target resource.
  - **TRACE** - Performs a message loop-back test along the path to the target resource.

Uniform Resource Location

- A **Uniform Resource Locator** (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a network and a mechanism for retrieving it.
- Is a specific type of Uniform Resource Identifier (URI).
- URLs occur most commonly to reference web pages (*http*) but are also used for file transfer (*ftp*), email (*mailto*), database access (*JDBC*), and many other applications.

Uniform Resource Location

- URL syntax



```
protocol://address/path/filename
```

- Examples:
  - http://www.upskill.pt
  - https://126.35.101.2:8080/examples/web/example1.html
  - ftp://myserver.pt/

UPskill

Digital Skills & Jobs

Web Development Evolution

Programação Orientada a Objetos

P.PORTO

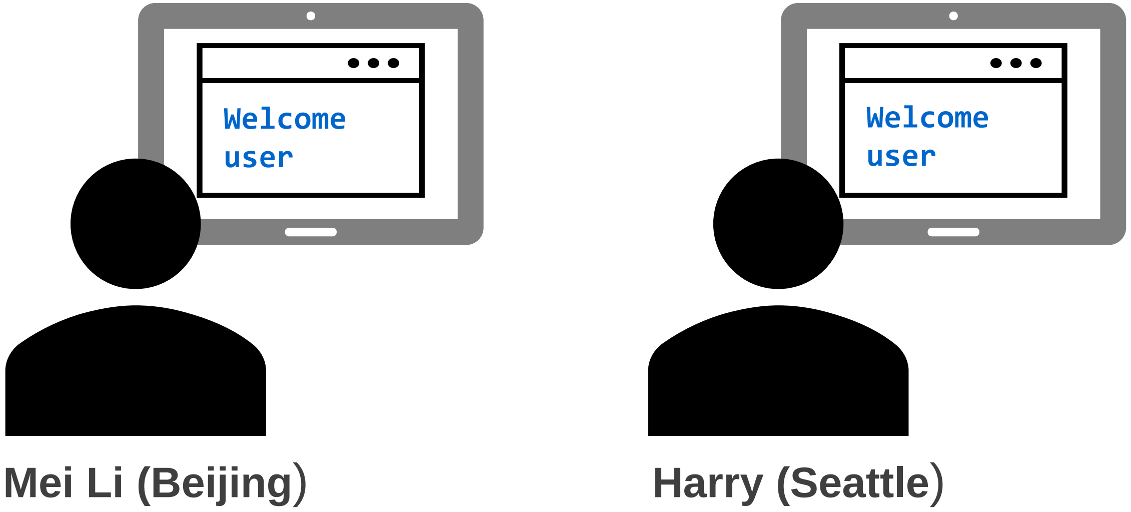
apdc
ASSOCIAÇÃO PORTUGUESA
PARA O DESENVOLVIMENTO
DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL

CONSELHO
COORDENADOR
DOS
INSTITUTOS
SUPERIORES
POLITECNICOS

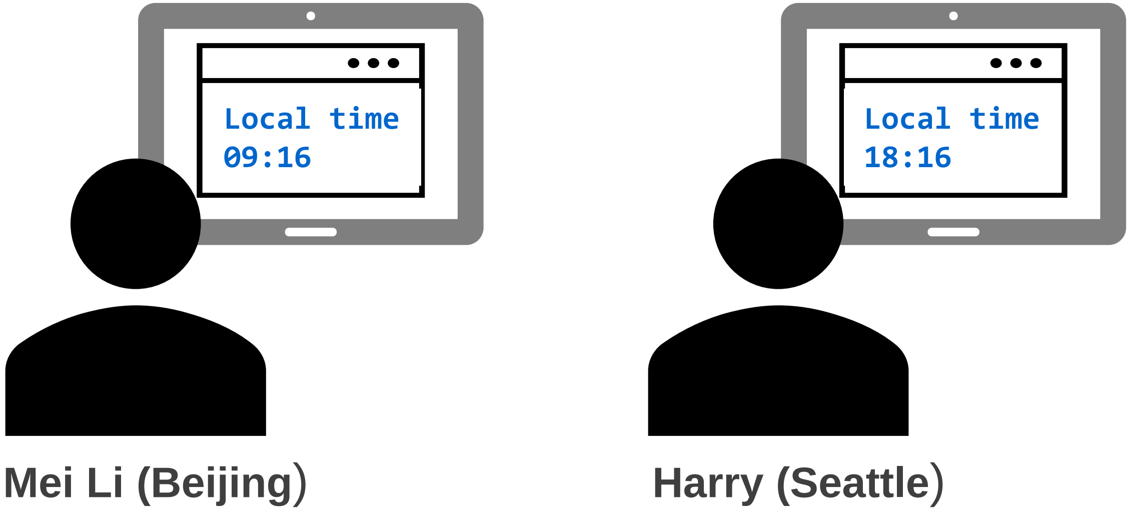
Static web pages

- The web server is simply a repository of pages (HTML files) that are sent at the client's request.
- The information and appearance of the page will be constant over time, changing only if the programmer intervenes on the content of the page.



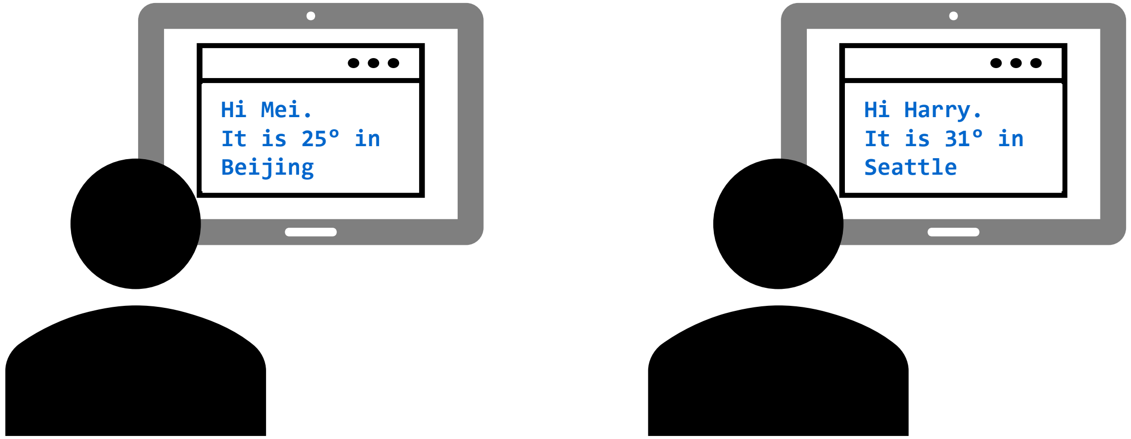
Static web pages with embedded scripts

- They are still static web pages.
- Dynamic objects are introduced on the page using a scripting language. These scripts are interpreted and executed in the client browser.



Dynamic web pages

- The server builds in real time the web pages.
- Allow interaction with external resources (e.g., databases).
- The appearance and information of the resulting page depends on parameters that are passed to the server with the request.



Hi Mei.
It is 25° in Beijing

Hi Harry.
It is 31° in Seattle

Mei Li (Beijing)

Harry (Seattle)

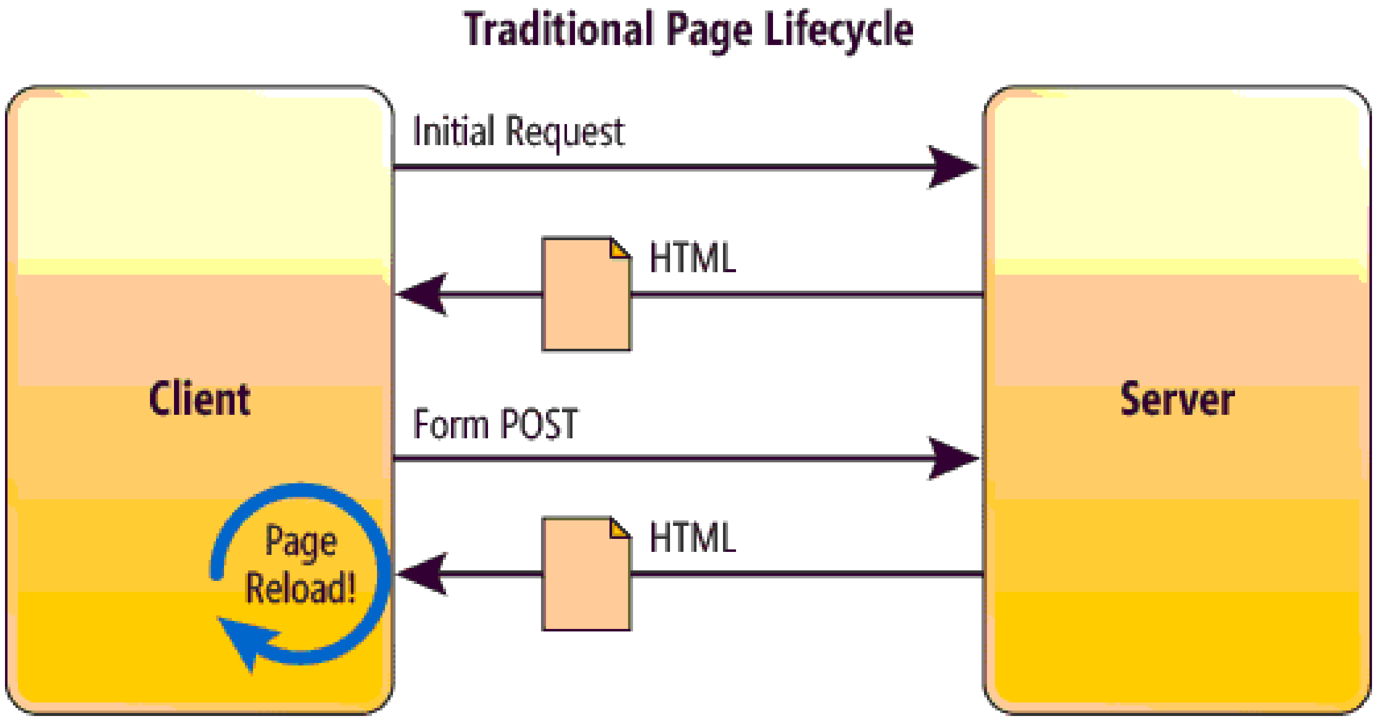
Dynamic web pages - Evolution

- Use of CGI (Common Gateway Interface) applications.
- There is an application on the server (CGI application) that will be invoked by the client's request and that will build in real time the page that will be sent to the client.
- CGI applications can be programmed in any language that accesses STDIN and STDOUT. The most common are:
  - *Perl*
  - *TCL*
  - *C*

Dynamic web pages - Evolution

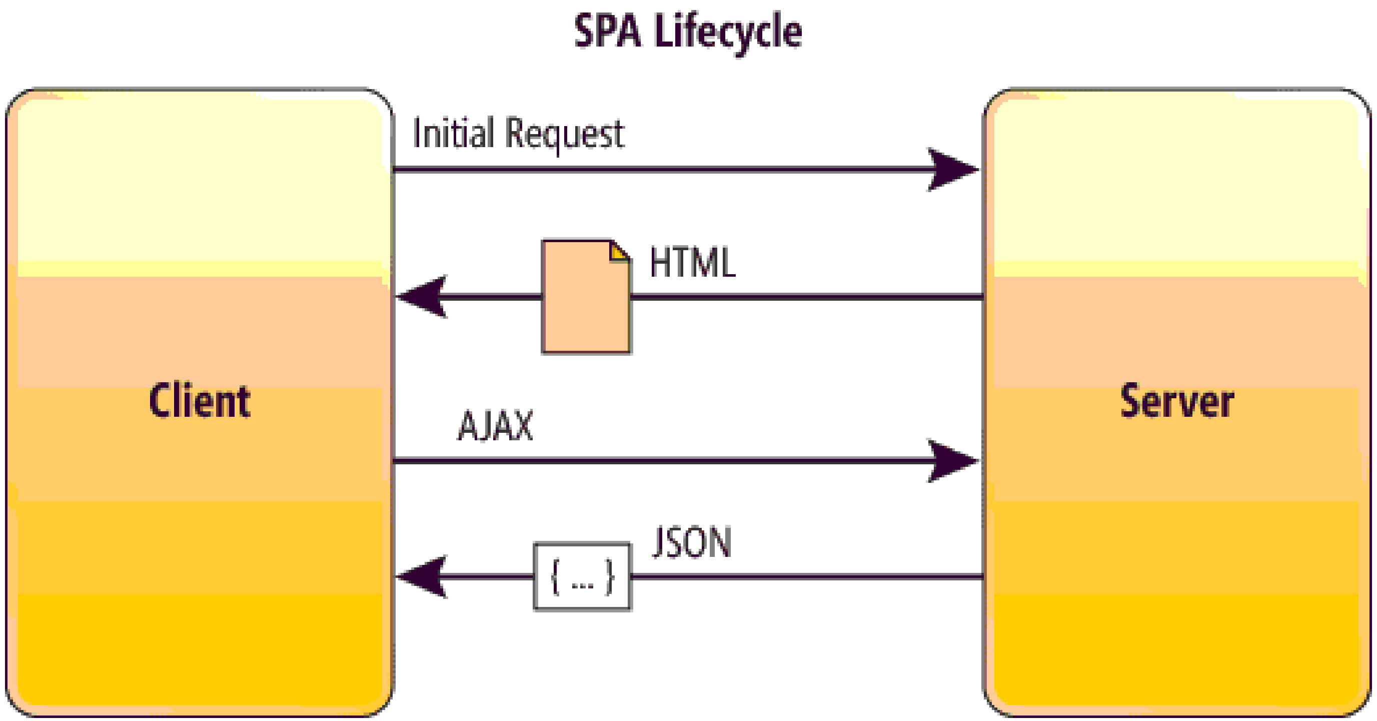
- Use of embedded code in the page’s source code.
- The script code used to generate the page coexists with the HTML commands that format the page.
- They are simpler to develop and maintain than CGI applications.
- The most common technologies in this category are:
  - ASP
  - PHP
  - Python
  - Javascript

Dynamic web pages – Traditional scenario



cf: https://moz.com/blog/optimizing-angularjs-single-page-applications-googlebot-crawlers

Dynamic web pages – SPA scenario



cf: https://moz.com/blog/optimizing-angularjs-single-page-applications-googlebot-crawlers

UPskill

Digital Skills & Jobs

Web Client-Server Architecture

Programação Orientada a Objetos

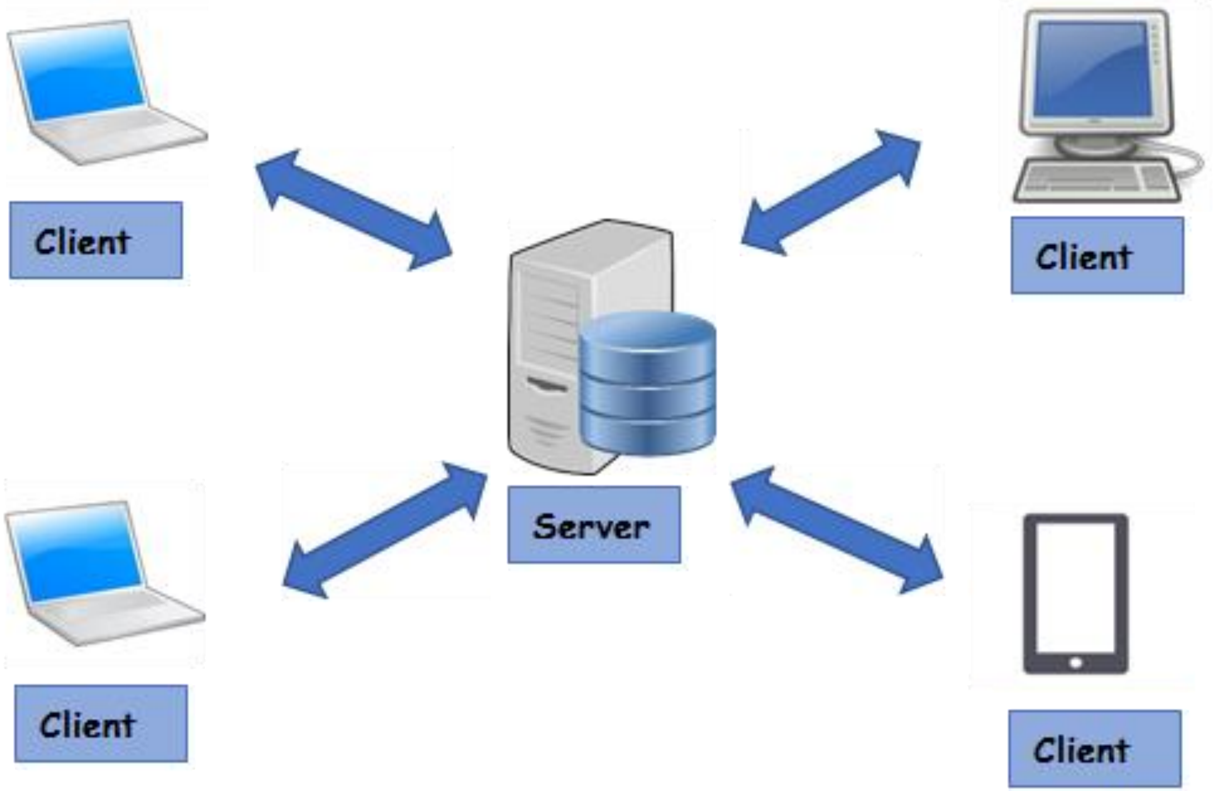
P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

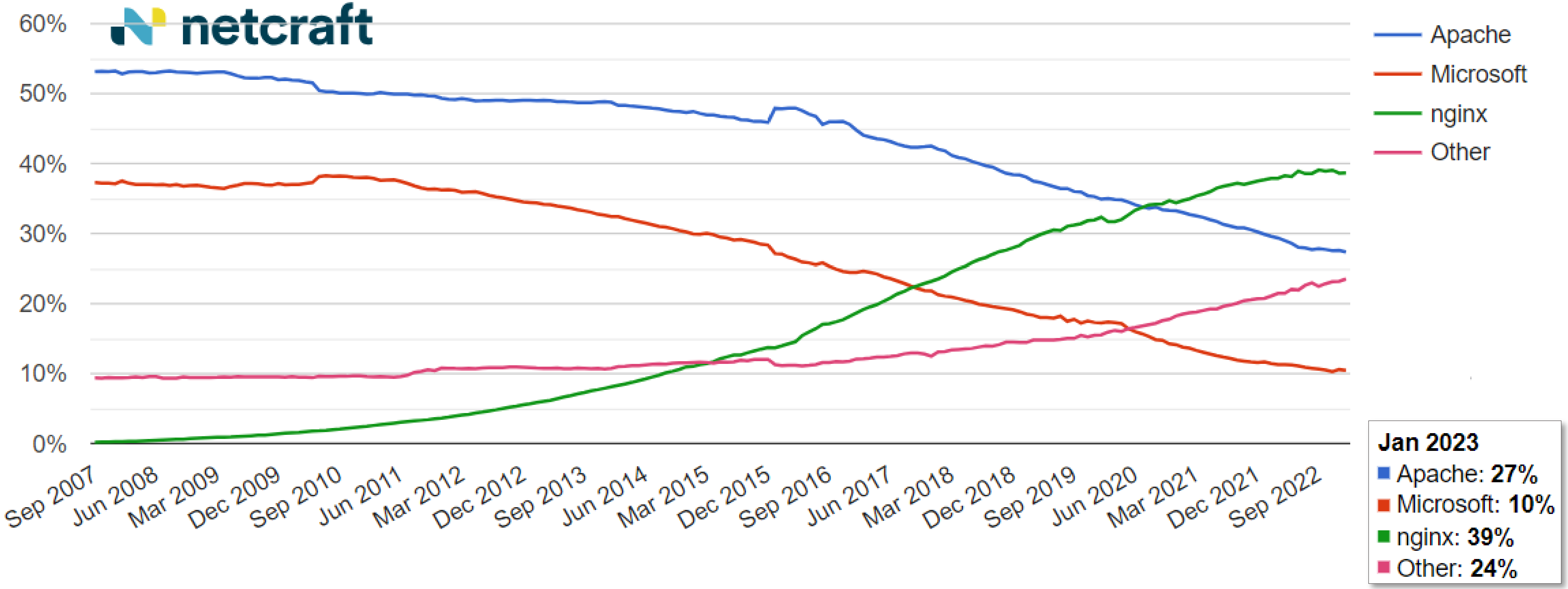
Web architecture



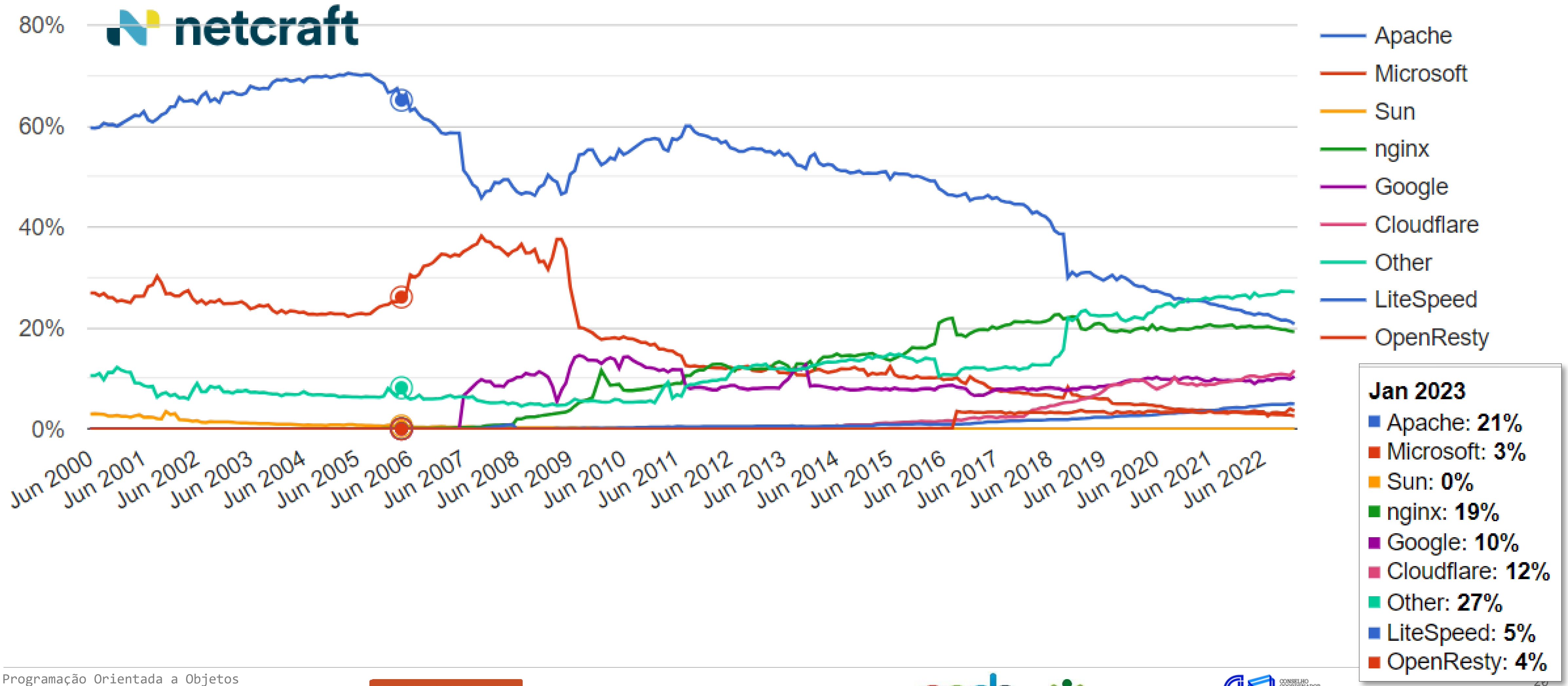
Web Server

- Computer software and underlying hardware that accepts requests via HTTP or its secure variant HTTPS.
- Is used to store and to deliver the website content.
- Most common servers
  - IIS - Internet Information Server
    - Most usual in the Windows universe
    - Developed by Microsoft
  - Apache
    - Usual on Unix/Linux systems but also exists for Windows systems
    - Free and open-source software
  - Nginx

Web Server – Market share of computers



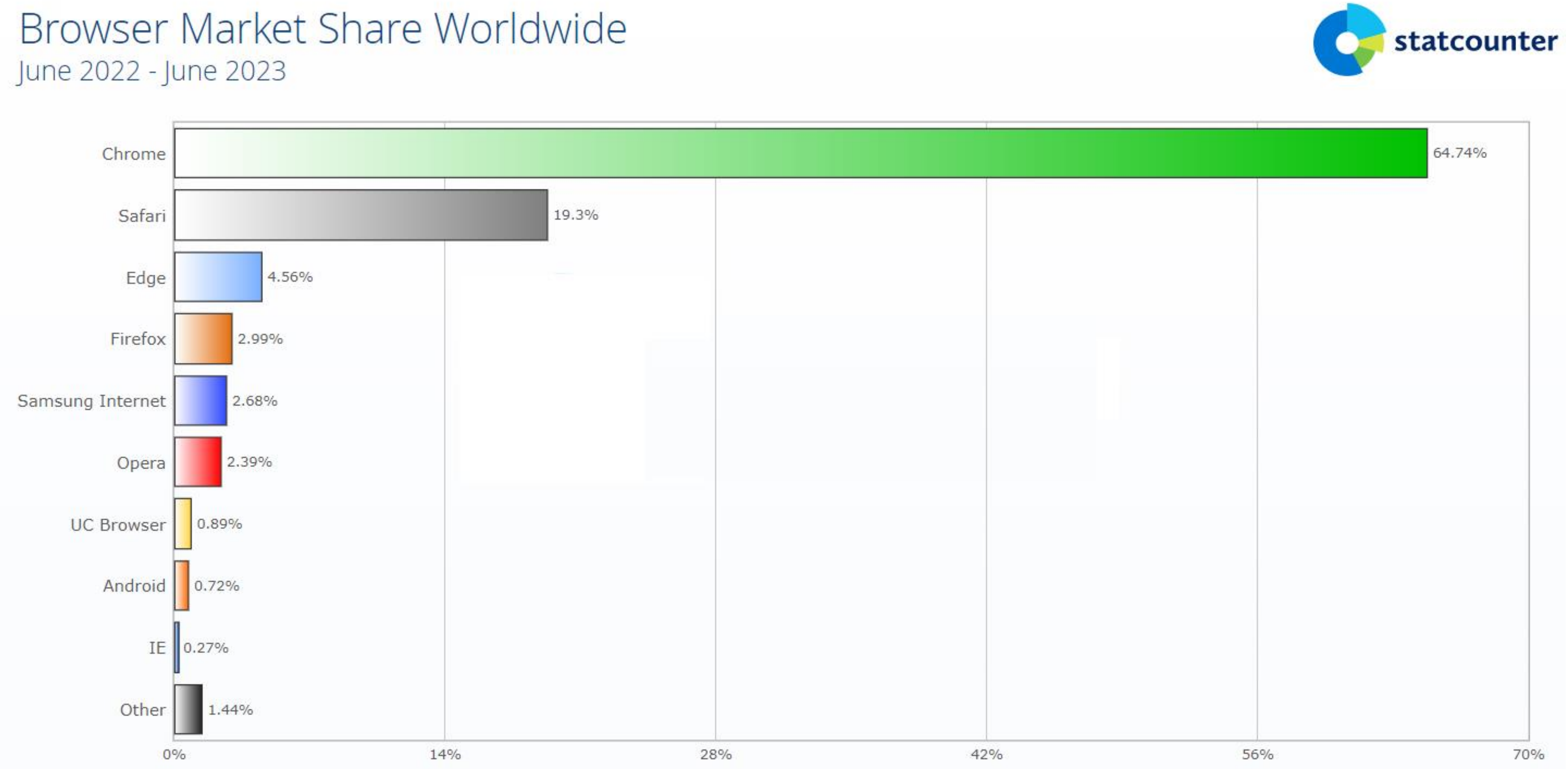
Web Server – Market share of active sites



Web Client

- Software application such as a Web browser
- Access and display web resources identified by a URL
  - Web pages
  - Images
  - Videos
- Process:
  - HTML, CSS, …
  - XML, JSON, …
  - SVG,
  - Scripting languages (e.g., javascript)

Web Client – Browser market share



Web Development - Summary

- Server-side
  - The information is processed by the web server and sent to the client for final processing and presentation.
  - Languages and Technologies:
    - PHP, .NET (C#, VB, F#), Node.js
    - Angular, ReactJS, VueJS
- Client-side
  - The web resource is interpreted for presentation by the browser
  - Languages and frameworks:
    - HTML, CSS, JavaScript, jQuery

UPskill

Digital Skills & Jobs

JSON

Programação Orientada a Objetos

P.PORTO

ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

What is JSON?

- Acronym for **J**ava**S**cript **O**bject **N**otation
- Is a lightweight data-interchange format
- Is self-describing and easy to understand
- Is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, (C, C++, C#, Java, JavaScript, Python, …)

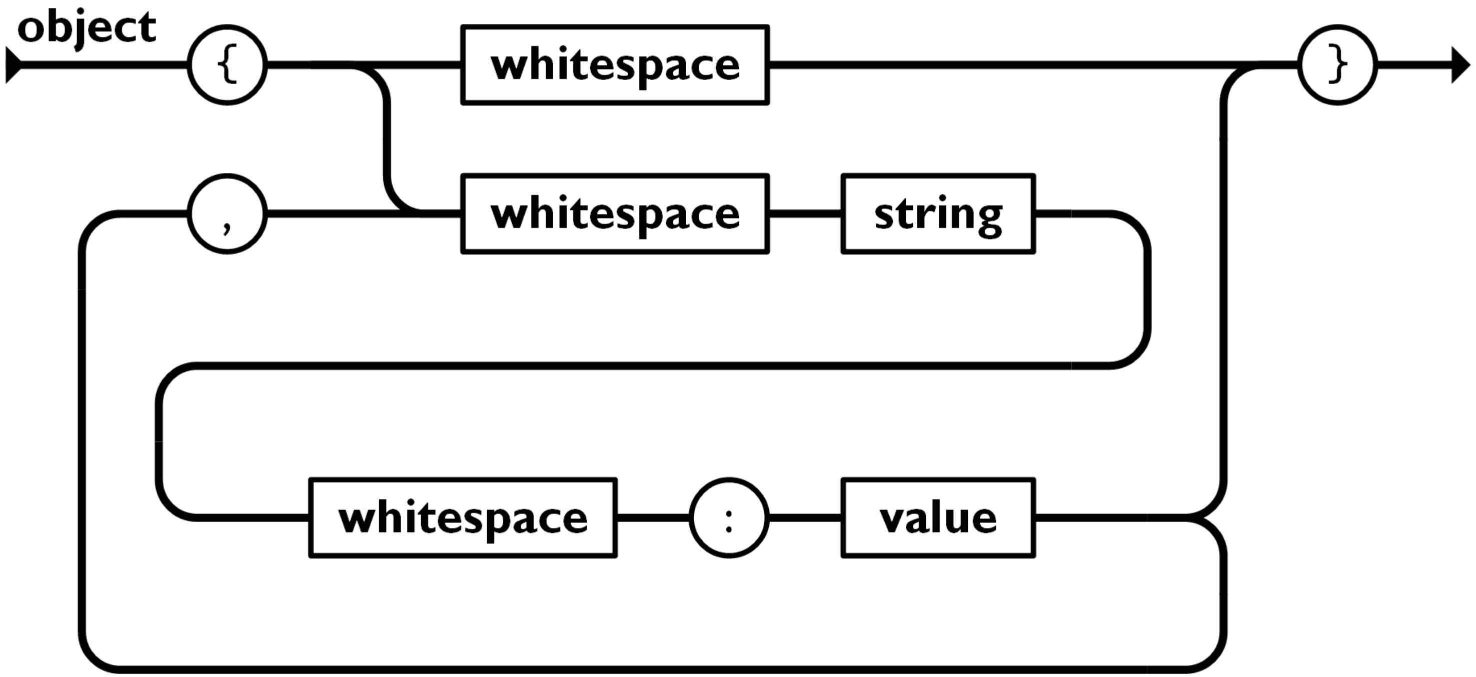
These properties make JSON an ideal data-interchange language

What is JSON?

- JSON is built on two structures:
  - A collection of name/value pairs.
  - An ordered list of values.

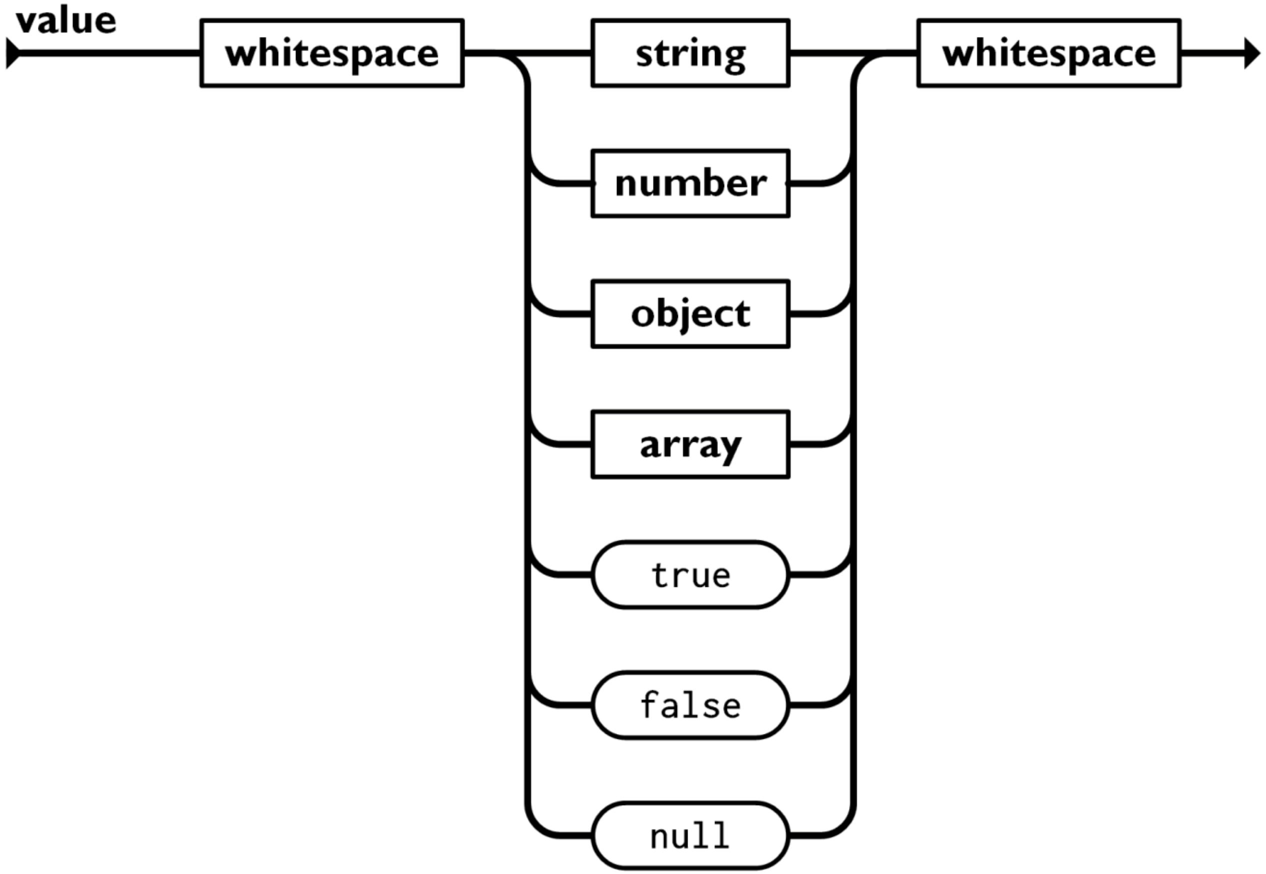
- These are universal data structures.
  - Virtually all modern programming languages support them in one form or another.
  - It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

JSON - object



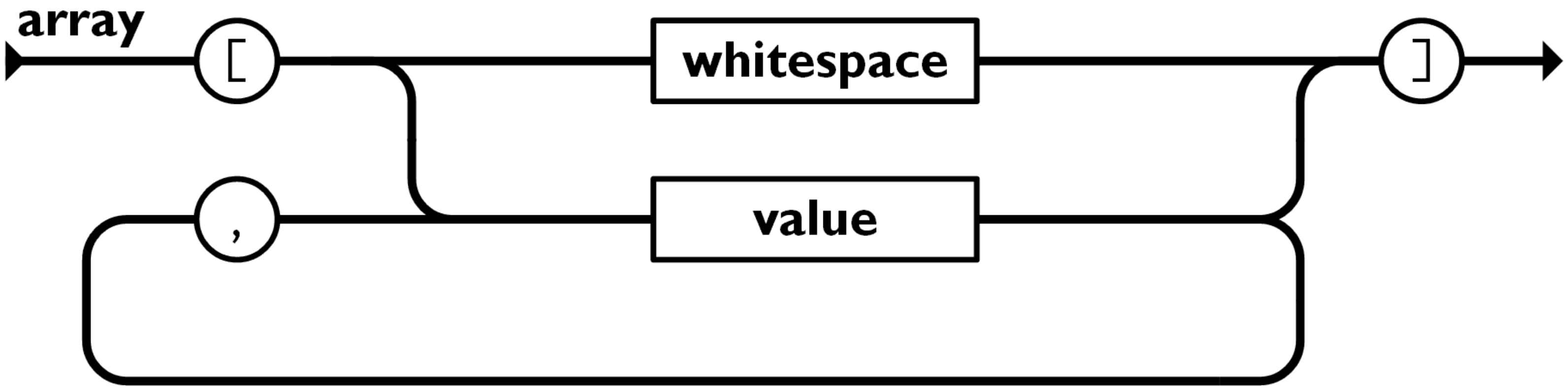
```
{
    "name": "John",
    "age": 31,
    "city": "New York"
}
```

JSON - value



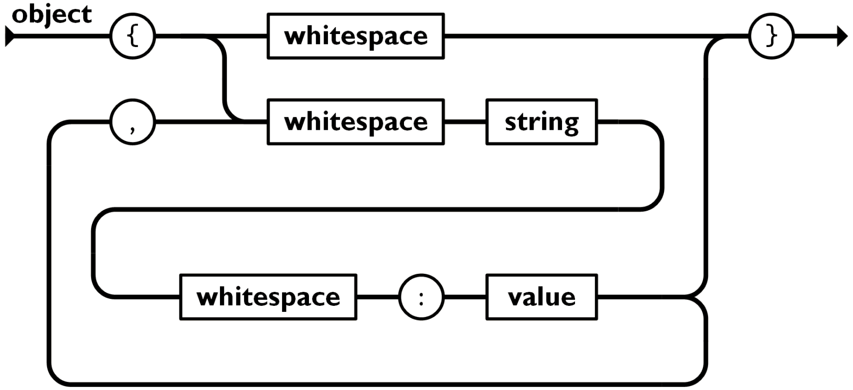
```
{
    "name": "John",
    "age": 31,
    "city": "New York",
    "canDrive":true
}
```

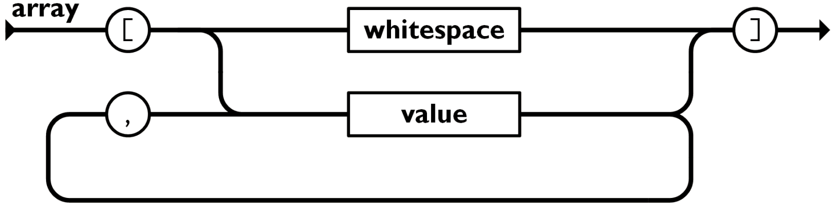
JSON - array



```
{
   "name": "John",
   "age": 31,
   "city": "New York",
   "canDrive":true,
   "cars":[
           "Ford",
           "BMW",
           "Fiat"
          ]
}
```

JSON - array





```
{
  "name":"John",
  "age":31,
  "city": "New York",
  "canDrive":true,
  "cars": [
    { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
    { "name":"BMW", "models":[ "320", "X3", "X5" ] },
    { "name":"Fiat", "models":[ "500", "Panda" ] }
  ]
}
```

UPskill

Digital Skills & Jobs

REST

Desenvolvimento de Aplicações Web

P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

What is REST?

- Acronym for REpresentational State Transfer
- It was first introduced by Roy Fielding in 2000

Fielding, R. T.; Taylor, R. N. (2000). "Principled design of the modern Web architecture". Proceedings of the 22nd international conference on Software engineering - ICSE '00. pp. 407–416.

Principled Design of the Modern Web Architecture

Roy T. Fielding and Richard N. Taylor
Information and Computer Science
University of California, Irvine
Irvine, CA 92697–3425 USA
+1.949.824.4121
{fielding,taylor}@ics.uci.edu

ABSTRACT

The World Wide Web has succeeded in large part because its software architecture has been designed to meet the needs of an *Internet-scale* distributed hypermedia system. The modern Web architecture emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. In this paper, we introduce the Representational State Transfer (REST) architectural style, developed as an abstract model of the Web architecture to guide our redesign and definition of the Hypertext Transfer Protocol and Uniform Resource Identifiers. We describe the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, contrasting them to the constraints of other architectural styles. We then compare the abstract model to the currently deployed Web architecture in order to elicit mismatches between the existing protocols and the applications they are intended to support.

Keywords

software architecture, software architectural style, WWW

1 INTRODUCTION

At the beginning of our efforts within the Internet Engineering Taskforce to define the existing Hypertext Transfer Protocol (HTTP/1.0) [5] and design the extensions for the new standards of HTTP/1.1 [10] and Uniform Resource Identifiers (URI) [6], we recognized the need for a model of how the World Wide Web (WWW, or simply Web) *should* work. This idealized model of the interactions within an overall Web application, what we refer to as the Representational State Transfer (REST) architectural style, became the foundation for the modern Web architecture, providing the guiding principles by which flaws in the preexisting architecture could be identified and extensions validated prior to deployment.

A software architecture determines how system elements are identified and allocated, how the elements interact to form a system, the amount and granularity of communication needed for interaction, and the interface protocols used for communication. An architectural style is an abstraction of the key aspects within a set of potential architectures (instantiations of the style), encapsulating important decisions about the architectural elements and emphasizing constraints on the elements and their relationships [17]. In other words, a style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to the style.

REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations. This is achieved by placing constraints on connector semantics where other styles have focused on component semantics. REST enables the caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, thereby meeting the needs of an Internet-scale distributed hypermedia system.

The modern Web is one instance of a REST-style architecture. Although Web-based applications can include access to other styles of interaction, the central focus of its protocol and performance concerns is distributed hypermedia. REST elaborates only those portions of the architecture that are considered essential for Internet-scale distributed hypermedia interaction. Areas for improvement of the Web architecture can be seen where existing protocols fail to express all of the potential semantics for component interaction, and where the details of syntax can be replaced with more efficient forms without changing the architecture capabilities. Likewise, proposed extensions can be compared to REST to see if they fit within the architecture; if not, it is more efficient to redirect that functionality to a system running in parallel with a more applicable architectural style.

This paper presents REST after the completion of six years' work on architectural standards for the modern (post-1993) Web. It does not present the details of the architecture itself, since those are found within the standards. Instead, we focus

What is REST?

- It is architectural style for distributed hypermedia systems
  - Not a specification…
- Enables the development of web services according to a specification
  - REST/HTTP is the most common instantiation.
- Two main ideas/assumptions
  1. Everything is a resource
  2. Each resource has a uniform interface

Key principles

1. Provide multiple representations
    - e.g., XML, JSON, XHTML, …

2. Give every “*thing*” and ID (Resources ID)

3. Use standard methods
    - e.g., HTTP methods
    - Communicate statelessly (no server session)

4. Link things together (HATEOAS)

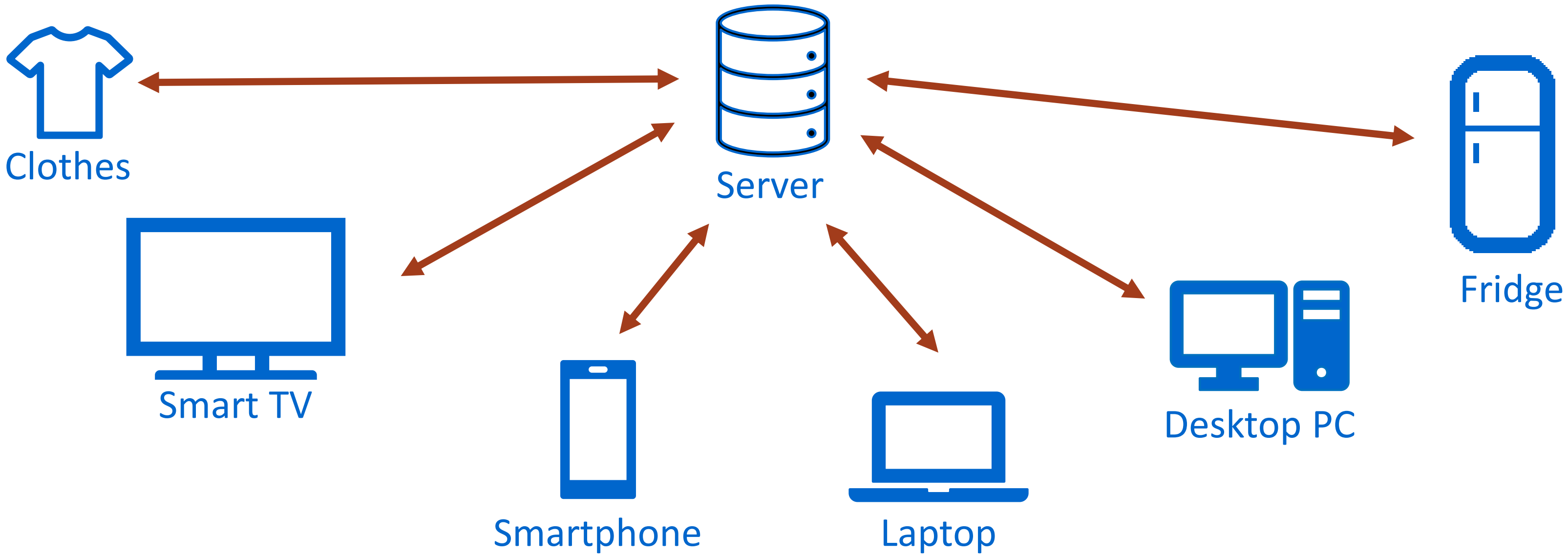
HTTP REST

- REST web services communicate over the HTTP specification, using HTTP vocabulary
  - Methods (GET, POST, PUT, DELETE, ...)
  - HTTP URI syntax (paths, parameters, ...)
  - Media types (xml, json, html, plain text, ...)
  - HTTP Response codes

Guiding Principles of REST

1. Client–server

By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.



Guiding Principles of REST

2. Stateless

   Each request from client to server must contain all the information necessary to understand the request and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

3. Cacheable

   Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

Guiding Principles of REST

4. Uniform interface

By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified, and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and hypermedia as the engine of application state.

Guiding Principles of REST

5. Layered system

   The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

6. Code on demand (optional)

   REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented

Guiding Principles of REST

- The six characteristics of REST
  - Decoupled client-server interaction
  - Stateless
  - Cacheable
  - Uniform interface
  - Layered system
  - Extensible through code on demand (optional)
- Services that conform to the above features are strictly RESTful web services

Resource

- The key abstraction of information in REST is a **resource**.
- Any information that can be named can be a resource:
  - a document
  - a image,
  - ….
- REST uses a **resource identifier** to identify the resource involved in an interaction between components.

Resource

- The state of the resource at any timestamp is known **as resource representation**.
- A representation consists of:
  - data
  - metadata describing the data
  - hypermedia links

  which can help the clients in transition to the next desired state.

Resource

- The data format of a representation is known as a **media type**.
- The media type identifies a specification that defines how a representation is to be processed.
- A truly RESTful API looks like hypertext.
- Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

Resource

- Resource representations shall be self-descriptive.
- It should act based on media-type associated with the resource.
- In practice, many custom media types will be created
  - usually, a media type associated with a resource.

Resource Methods

- RESTful web service makes use of HTTP for determining the action to be conceded out on the resources
- The primary or most-commonly-used HTTP verbs (methods) are **POST**, **GET**, **PUT**, **PATCH**, and **DELETE**.
- These correspond to create, read, update, and delete (or CRUD) operations

Verb, or HTTP method meaning

Post	Create
Get	Read
Put	Update
Patch	Partial Update
Delete	Delete

Head	Return header of HTTP
Options	Returns the operations available in the Resource

Verb, or HTTP method meaning

- GET – to read a resource
  - Is used to read a Resource or Resources.
  - Is considered safe, it should never modify the state of a resource.
- POST – to create a resource
  - Is used to create a Resource.
  - Resource values are sent to the server as part of the request body.
- PUT – to update a resource
  - Is used to update a Resource.
  - The URI specifies the Resource we want to modify and the body contains the new Resource values.
- DELETE – to delete a resource
  - Is used to delete/remove a Resource
  - The URI specifies the Resource

HTTP-REST Request Basics

- The HTTP request is sent from the client
  - Identifies the location of a resource
  - Specifies the verb, or HTTP method
    - To use when accessing the resource
  - Supplies optional request headers
    - name-value pairs
    - Provide additional information the server may need when processing the request
  - Supplies an optional request body
    - That identifies additional data to be uploaded to the server (e.g. form parameters, attachments, etc.)

HTTP-REST Response Basics

- The HTTP response is sent from the server
  - Gives the status of the processed request
  - Supplies response headers (name-value pairs) that provide additional information about the response
  - Supplies an optional response body that identifies additional data to be downloaded to the client (html, xml, binary data, etc.)

HTTP response status codes

- HTTP response status codes indicate whether a specific HTTP request has been successfully completed.

- Responses are grouped in five classes:

1	Informational responses	100 - 199
2	Successful responses	200 - 299
3	Redirects	300 - 399
4	Client errors	400 - 499
5	Server errors	500 - 599

HTTP response status codes

- Information responses
  - `100 Continue`

    This interim response indicates that everything so far is OK and that the client should continue the request or ignore the response if the request is already finished.
  - `101 Switching Protocol`

    This code is sent in response to an Upgrade request header from the client and indicates the protocol the server is switching to.
  - `102 Processing (WebDAV)`

    This code indicates that the server has received and is processing the request, but no response is available yet.

HTTP response status codes

- Successful responses
  - `200 OK`

    The request has succeeded. The meaning of the success depends on the HTTP method:

    - GET: The resource has been fetched and is transmitted in the message body.
    - HEAD: The entity headers are in the message body.
    - PUT or POST: The resource describing the result of the action is transmitted in the message body.
    - TRACE: The message body contains the request message as received by the server.
  - `201 Created`

    The request has succeeded, and a new resource has been created as a result. This is typically the response sent after POST requests, or some PUT requests

HTTP response status codes

- Successful responses
  - `202 Accepted`

    The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.
  - `203 Non-Authoritative Information`

    This response code means the returned meta-information is not exactly the same as is available from the origin server but is collected from a local or a third-party copy. Except for that specific case, the "200 OK" response is preferred to this status.
  - `204 No Content`

    There is no content to send for this request, but the headers may be useful. The user-agent may update its cached headers for this resource with the new ones.

HTTP response status codes

- Successful responses
  - `205 Reset Content`

    Tells the user-agent to reset the document which sent this request.
  - `206 Partial Content`

    This response code is used when the Range header is sent from the client to request only part of a resource.

HTTP response status codes

- Redirection responses
  - `300 Multiple Choice`

    The request has more than one possible response. The user-agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)
  - `301 Moved Permanently`

    The URL of the requested resource has been changed permanently. The new URL is given in the response.
  - `302 Found`

    This response code means that the URI of requested resource has been changed temporarily. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests

HTTP response status codes

- Redirection responses
  - `303 See Other`

    The server sent this response to direct the client to get the requested resource at another URI with a GET request.
  - `304 Not Modified`

    This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.
  - `305 Use Proxy`

    Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. Deprecated due to security concerns regarding in-band configuration of a proxy.
  - `306 unused`

    This response code is no longer used; it is just reserved. It was used in a previous version of the HTTP/1.1 specification

HTTP response status codes

- Redirection responses
  - `307 Temporary Redirect`

    The server sends this response to direct the client to get the requested resource at another URI with same method that was used in the prior request. This has the same semantics as the 302 Found HTTP response code, with the exception that the user agent must not change the HTTP method used: If a POST was used in the first request, a POST must be used in the second request.
  - `308 Permanent Redirect`

    This means that the resource is now permanently located at another URI, specified by the Location: HTTP Response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent must not change the HTTP method used: If a POST was used in the first request, a POST must be used in the second request.

HTTP response status codes

- Client error responses
  - `400 Bad Request`

    The server could not understand the request due to invalid syntax.
  - `401 Unauthorized`

    Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.
  - `402 Payment Required`

    This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems.
  - `403 Forbidden`

    The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike `401`, the client's identity is known to the server.

HTTP response status codes

- Client error responses
  - `404 Not Found`

    The server can not find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of `403` to hide the existence of a resource from an unauthorized client. This response code is probably the most famous one due to its frequent occurrence on the web.
  - `406 Not Acceptable`

    This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent.
  - `407 Proxy Authentication Required`

    This is similar to `401` but authentication is needed to be done by a proxy

HTTP response status codes

- Client error responses
  - `408 Request Timeout`

    This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also note that some servers merely shut down the connection without sending this message.
  - `414 URI Too Long`

    The URI requested by the client is longer than the server is willing to interpret.
  - `422 Unprocessable Entity`

    The request was well-formed but was unable to be followed due to semantic errors.
  - `429 Too Many Requests`

    The user has sent too many requests in a given amount of time ("rate limiting").

HTTP response status codes

- Server error responses
  - `500 Internal Server Error`

    The server has encountered a situation it doesn't know how to handle.
  - `501 Not Implemented`

    The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.
  - `502 Bad Gateway`

    This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

HTTP response status codes

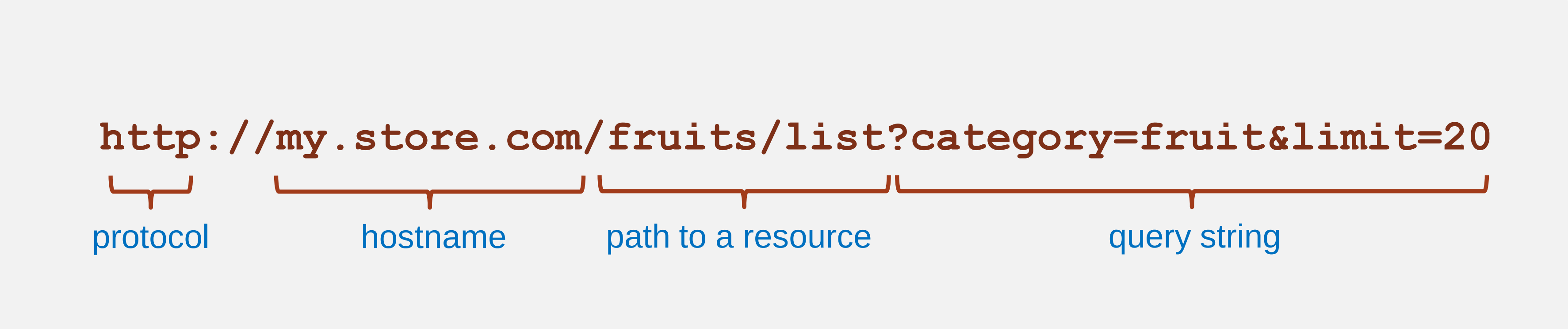
- Server error responses
  - `503 Service Unavailable`

    The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This responses should be used for temporary conditions and the Retry-After: HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.
  - `504 Gateway Timeout`

    This error response is given when the server is acting as a gateway and cannot get a response in time

REST URI

- A typical HTTP REST URL:



```
http://my.store.com/fruits/list?category=fruit&limit=20
```

- The protocol identifies the transport scheme that will be used to process and respond to the request
- The host name identifies the server address of the resource.
- The path and query string can be used to identify and customize the accessed resource

URI path design rules

- A singular noun should be used for document names
  - http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/claudio
- A plural noun should be used for collection names
  - http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players
- A plural noun should be used for store names
  - http://api.music.restapi.org/artists/mikemassedotcom/playlists
- A verb or verb phrase should be used for controller names
  - http://api.college.restapi.org/students/morgan/register
- Variable path segments may be substituted with identity-based values
  - http://api.soccer.restapi.org/leagues/{leagueId}/teams/{teamId}/players/{playerId}
- CRUD function names should not be used in URIs
  - `DELETE /users/12` (is preferred over) `GET /deleteUser/12` (or) `DELETE /deleteUser/12`

from: REST API Design Rulebook; Mark Massé; O'Reily; 2011

URI query design rules

- The query component of a URI may be used to filter collections or stores
  - `GET /users can be complemented` with `GET /users/?role=admin`
- The query component of a URI should be used to paginate collection or store results
  - `GET /users?pageSize=25&pageStartIndex=50`
  - `POST /users/search`, where the body includes more complex pagination info

from: REST API Design Rulebook; Mark Massé; O'Reily; 2011

Example/Exercises

US	Description	Method	Resource/URL	Input Body	Output Body	Status
US001	Como gestor de Sistema quero saber se determinada pessoa é irmão/irmã de outra.	GET	/pessoas/{id1}/irmaos/{id2}	n/a	{"isIrmao":true}	200/422
US002.1	Como utilizador, quero criar grupo, tornando-me administrador de grupo.	POST PUT	/grupos/	{"desc":"xpto", "idPessoaAdm":123, ...}	{"_self":"uri"}	201/422
			/pessoas/{id}/grupos/	{"desc":"xpto", ~~"idPessoaAdm":123,~~ ...}		

US003	Como gestor de sistema, quero acrescentar pessoas ao grupo.	POST(!)
		PATCH(?)
		PUT(?)
			/grupos/{id1}/membros/{id2}		{"_self":"uri"}	201/422

			/grupos/{id}/membros	[id2,id3,id4]	{"_self":"uri", {"sucess":[id2, id4]}, {"insucess":[id3]} }	
US004	Como gestor quero saber quais os grupos que são família, [...]	GET				

Some examples

http://www.xpto.com/magazines

http://www.xpto.com/magazines?year=2020&sort=desc

http://www.xpto.com/magazines/1234/articles

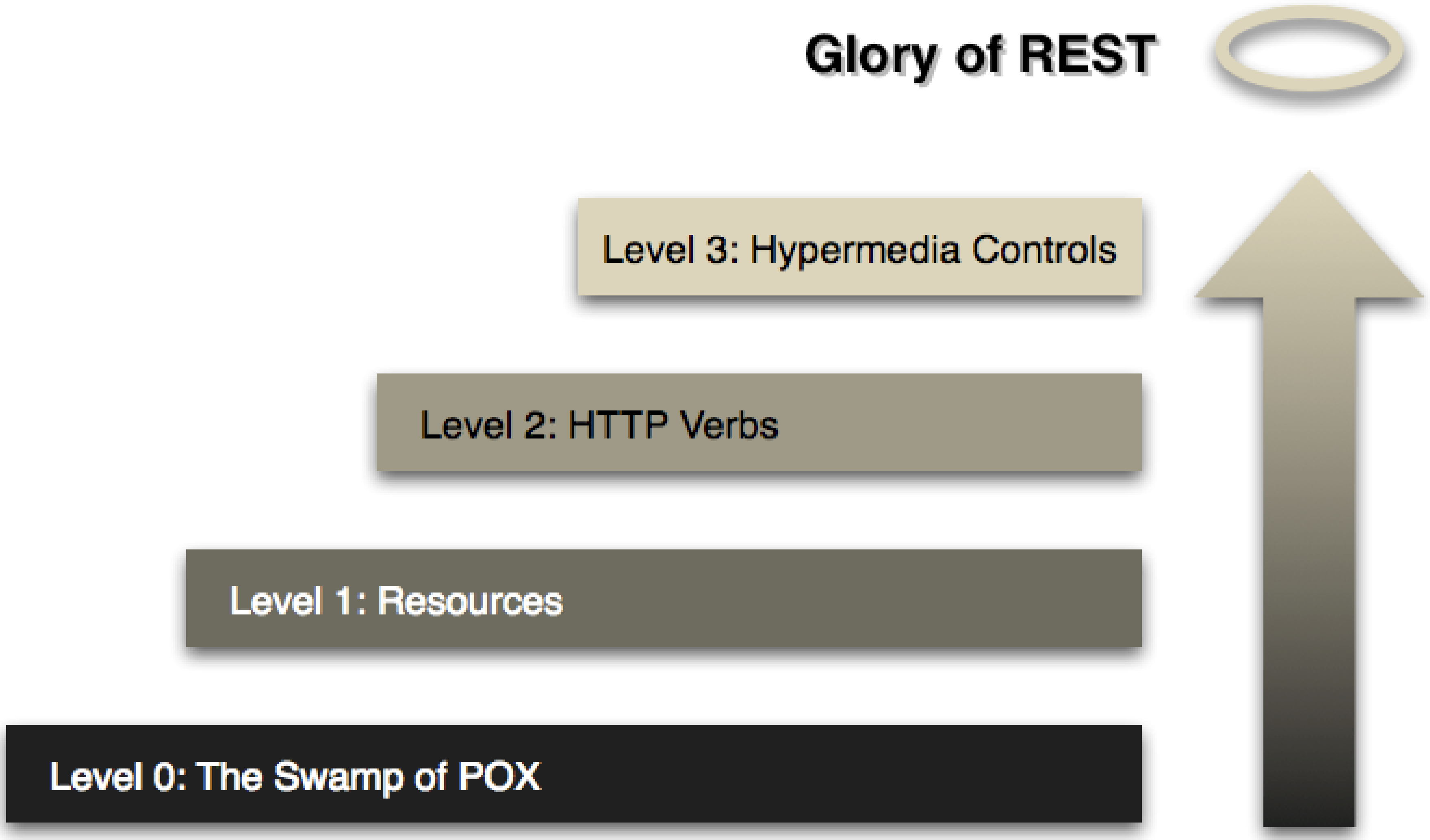
http://www.xpto.com/magazines/1234/articles?page=20

http://www.xpto.com/magazines/1234/articles/authors/Mary

Richardson Maturity Model

- Developed by Leonard Richardson
- A model that breaks down the principal elements of a REST approach into three steps. These introduce:
  - Resources
  - http verbs
  - hypermedia controls

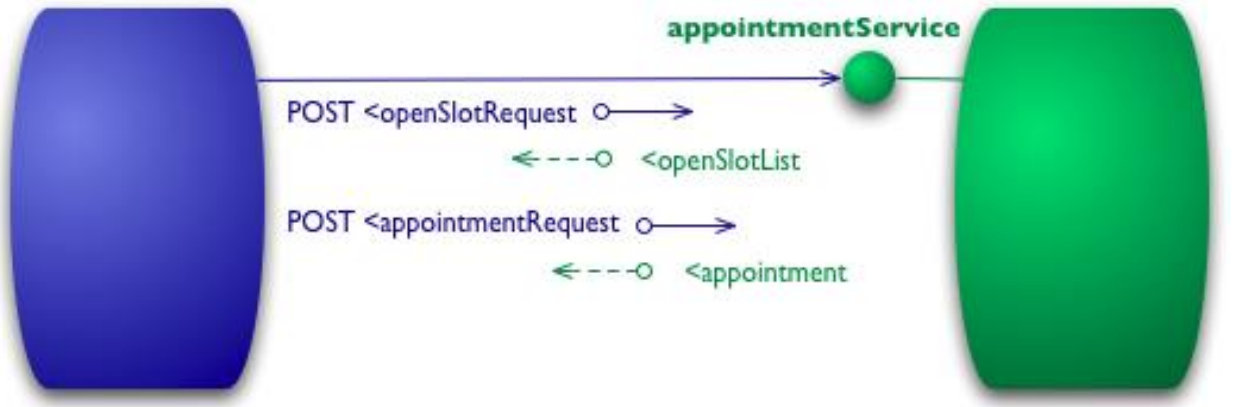
Richardson Maturity Model



Richardson Maturity Model

▪ Level 0 – The Swamp of POX (*Plain Old XML*)

- HTTP as a transport system for remote interactions, but without using any of the mechanisms of the web.
- Uses HTTP as a tunneling mechanism for its remote interaction mechanism, usually based on Remote Procedure Invocation.



Richardson Maturity Model

- **Level 0**

  Let's assume that a patient wants to make an appointment with his doctor. The scheduling application needs to know the doctor's availability on a given date, and for that it makes a request to the hospital's scheduling system to obtain this information.

  - In a level 0 scenario, the hospital will expose a service endpoint.
  - The patient will post to that endpoint a document containing the details of their request.
  - The server will return a document with the required information

Richardson Maturity Model

- **Level 0**

Availability Request:

```
POST /appointmentService HTTP/1.1
[headers]
[Body] {"date":"2023-07-08", "doctor":"mjones"}
```

Availability Response:

```
HTTP/1.1 200 OK
{"openSlotsList": [
          {"start":"14:00", "doctor":"mjones"},
          {"start":"14:45", "doctor":"mjones"}
        ]
}
```

Richardson Maturity Model

- **Level 0**

Appointment Request:

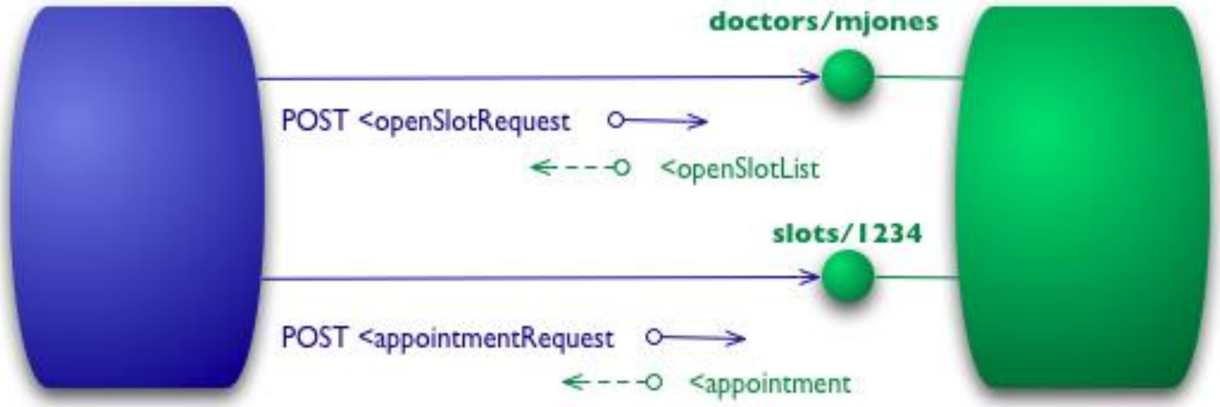
```
POST /appointmentService HTTP/1.1
[headers]
[Body] {"date":"2023-07-08", "doctor":"mjones", "patient"="jsmith"}
```

Appointment Response:

```
HTTP/1.1 200 OK
{"date":"2023-07-08", "doctor":"mjones", "patient"="jsmith"}
```

Richardson Maturity Model

- **Level 1 - Resources**
  - Talk to individual resources.



Richardson Maturity Model

- **Level 1**

Availability Request:

```
POST /doctors/mjones HTTP/1.1
[headers]
[Body] {"date":"2023-07-08"}
```

Availability Response:

```
HTTP/1.1 200 OK
{"openSlotsList": [
          {"id":"1234","start":"14:00", "doctor":"mjones"},
          {"id":"5678","start":"14:45", "doctor":"mjones"}
        ]
}
```

Richardson Maturity Model

- **Level 1**

Appointment Request:

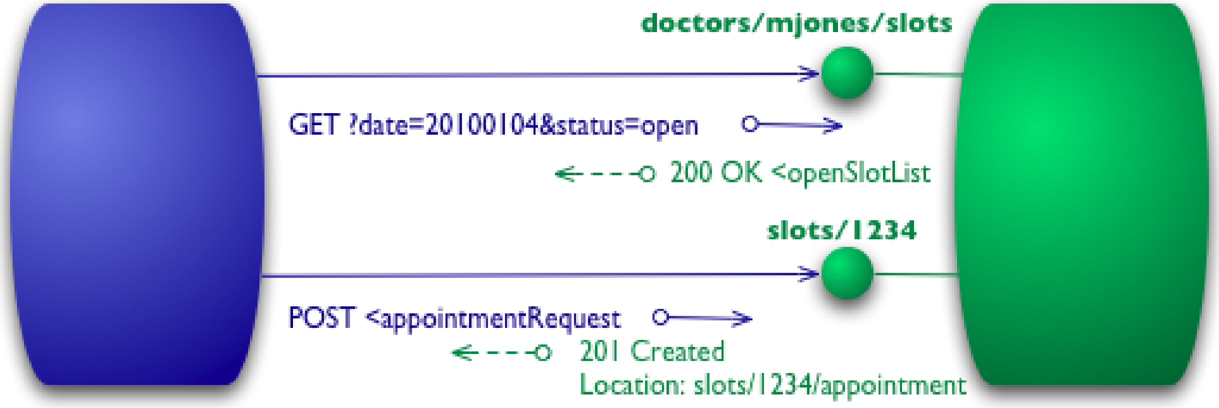
```
POST /slots/1234 HTTP/1.1
[headers]
[Body] {"patient":"jsmith"}
```

Appointment Response:

```
HTTP/1.1 200 OK
{"id":"1234","start":"14:00", "doctor":"mjones","patient":"jsmith"}
```

Richardson Maturity Model

- **Level 2**
  - Uses the HTTP verbs as closely as possible to how they are used in HTTP itself.



Richardson Maturity Model

- **Level 2**

Availability Request:

```
GET /doctors/mjones/slots?date=20230708&status=open HTTP/1.1
```

Availability Response:

```
HTTP/1.1 200 OK
{"openSlotsList": [
          {"id":"1234","start":"14:00", "doctor":"mjones"},
          {"id":"5678","start":"14:45", "doctor":"mjones"}
        ]
}
```

Richardson Maturity Model

- **Level 2**

Appointment Request:

```
POST /slots/1234 HTTP/1.1
[Body] {"patient":"jsmith"}
```

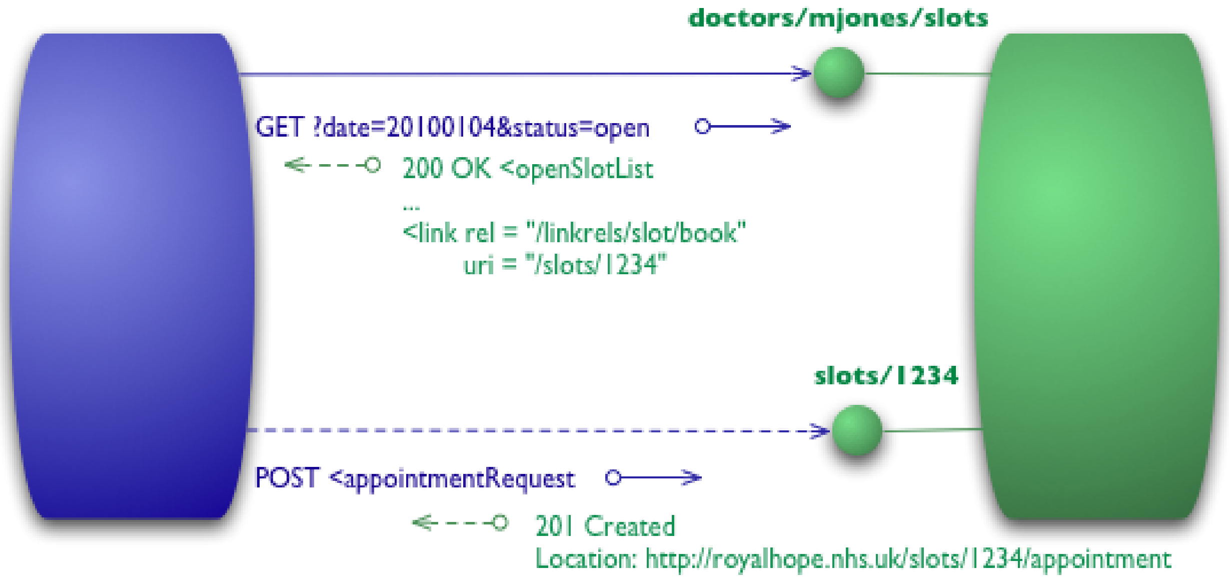
Appointment Response:

```
HTTP/1.1 201 Created
{"id":"1234","start":"14:00", "doctor":"mjones","patient":"jsmith"}
```

```
HTTP/1.1 409 Conflict
{"id":"1234","start":"14:00", "doctor":"mjones"}
```

Richardson Maturity Model

- **Level 3**
  - Hypermedia controls
  - Introduces **HATEOAS** (*Hypertext As The Engine Of Application State*)



doctors/mjones/slots

GET ?date=20100104&status=open

200 OK <openSlotList
...
<link rel = "/linkrels/slot/book"
uri = "/slots/1234"

slots/1234

POST <appointmentRequest

201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment

Hypermedia

- The fundamental idea of hypermedia is to enrich the representation of a resource with hypermedia elements.
- The simplest form of that are links (hypertext).
- They indicate a client that it can navigate to a certain resource.
- The semantics of a related resource are defined in a so-called link relation.

```
{
    "_links": {
        "self": {
            "href": "http://ex.com/customers/1234"
        }
    }
}
```

Richardson Maturity Model

- **Level 3**
  - It addresses the question of how to get from a list open slots to knowing what to do to book an appointment.

Availability Request:

```
GET /doctors/mjones/slots?date=20230708&status=open HTTP/1.1
```

Availability Response:

```
HTTP/1.1 200 OK
{"openSlotsList": [
          {"id":"1234","start":"14:00", "doctor":"mjones",
                               "_links":{"book":{"href":"/slots/1234"}}},
          {"id":"5678","start":"14:45", "doctor":"mjones",
                               "_links":{"book":{"href":"/slots/5678"}}}
        ]
}
```

Richardson Maturity Model

- **Level 3**

Appointment Request:

```
POST /slots/1234 HTTP/1.1
```

Availability Response:

```
HTTP/1.1 201 Created
{"id":"1234","start":"14:00", "doctor":"mjones","patient":"jsmith",
        "_links":{
                "self":{"href":"/slots/1234"},
                "cancel":{"href":"/slots/1234"},
                "updateTime":{"href":"/slots/1234/update"},
                "updateContact":{"href":"/patients/jsmith/contactInfo"}
        }
}
```

UPskill

Digital Skills & Jobs

Web Api

Programação Orientada a Objetos

P.PORTO

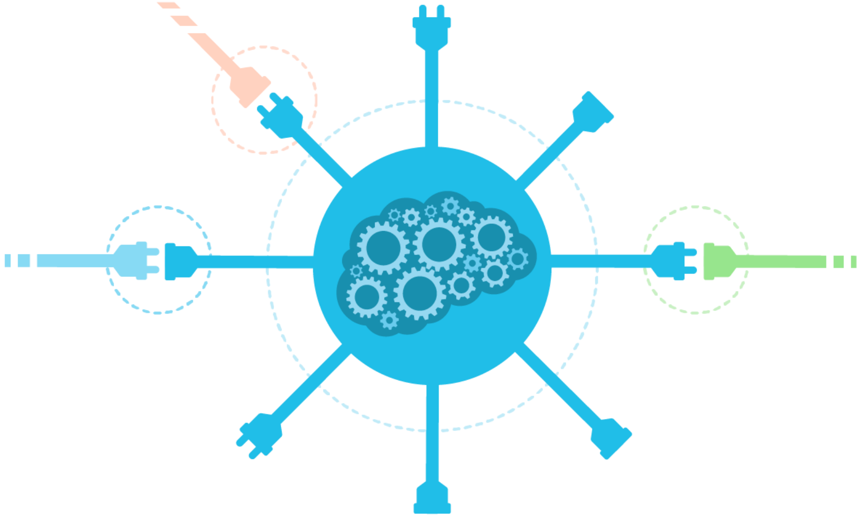
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

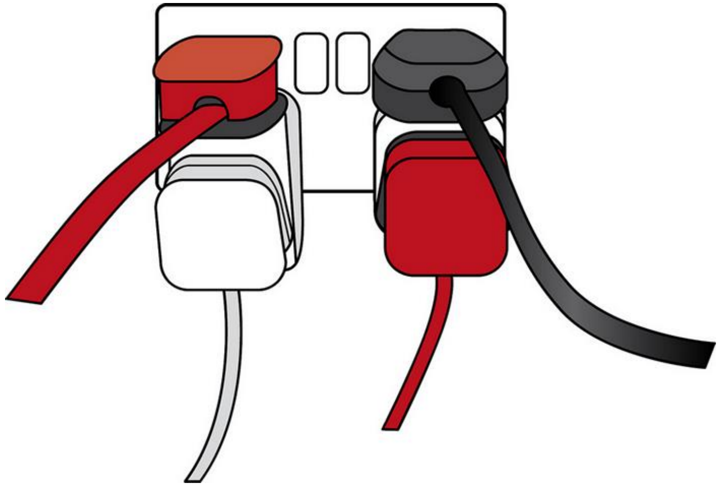
What is an API?

- **A**pplication **P**rogramming **I**nterface
- Interface with a set of functions, protocols, and tools that allow programmers to access specific features or data of a system for building new software applications

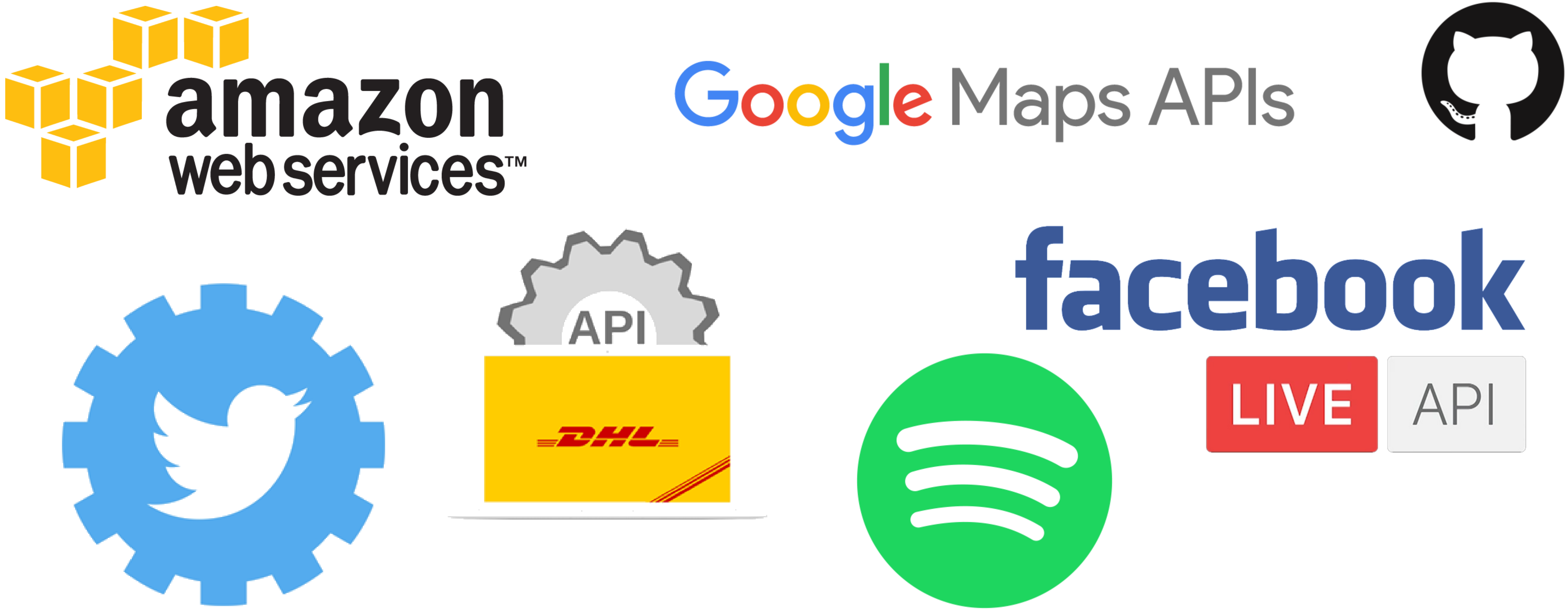


What is an API?

- Real-world example:
  - If you want to use an appliance in your house, you simply plug it into a plug socket, and it works. You don't try to wire it directly into the power supply — to do so would be really inefficient and, if you are not an electrician, difficult and dangerous to attempt.

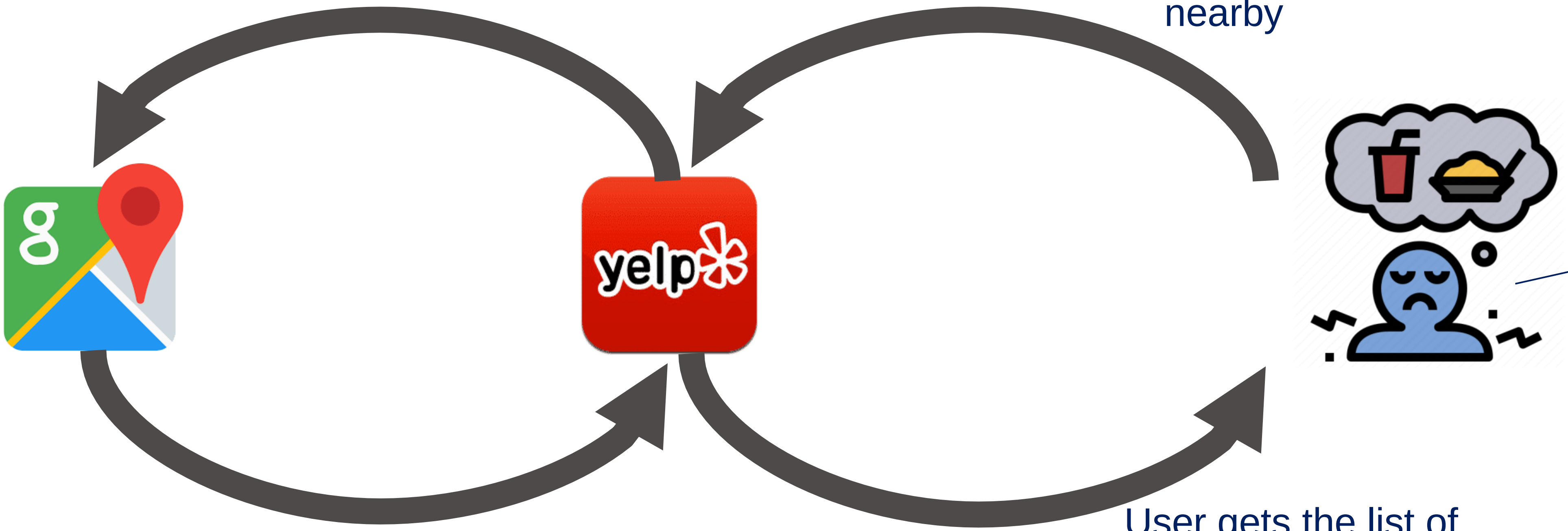


Why are API’s so important?



Why are APIs so important?

- API’s help developers create apps that benefit the end user



Yelp asks for Map Data

User wants to know restaurants nearby

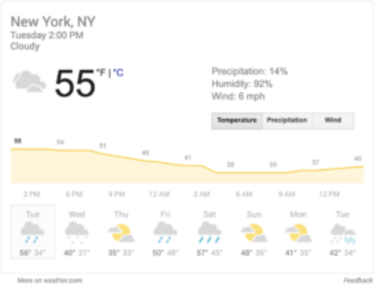
User is hungry.

Google Maps returns data via API

User gets the list of nearby restaurants

Examples of common APIs

- Weather Snippets

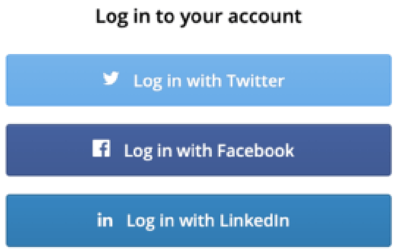


New York, NY
Tuesday 2:00 PM
Cloudy

55 °F | °C

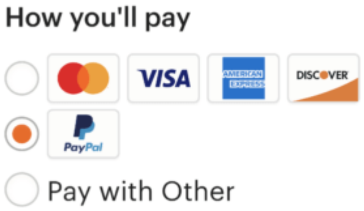
Precipitation: 14%
Humidity: 92%
Wind: 6 mph

- Log-in



Log in to your account

- Pay with PayPal



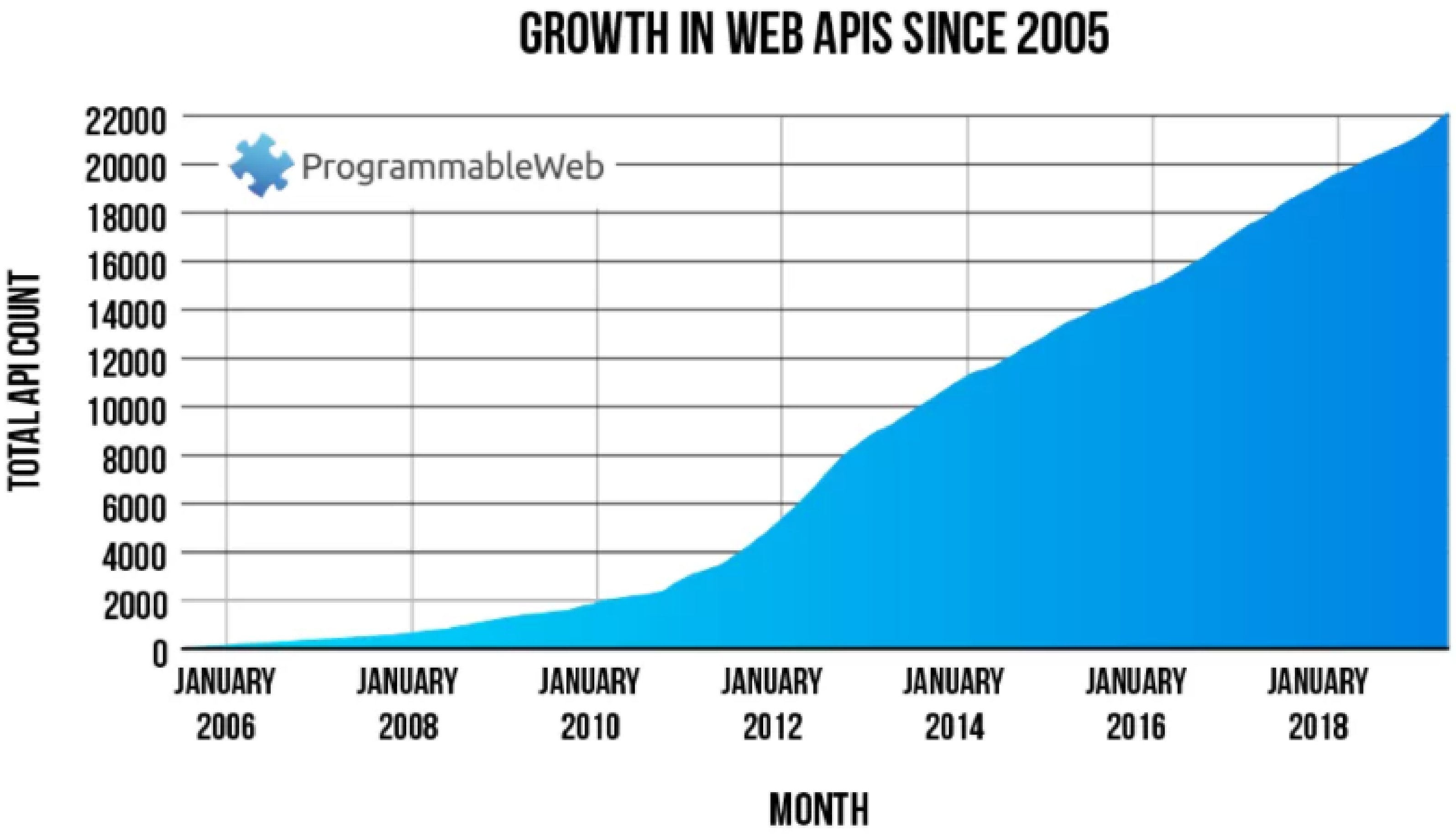
How you'll pay

○ MasterCard VISA AMERICAN EXPRESS DISCOVER

● PayPal

○ Pay with Other

API Growth

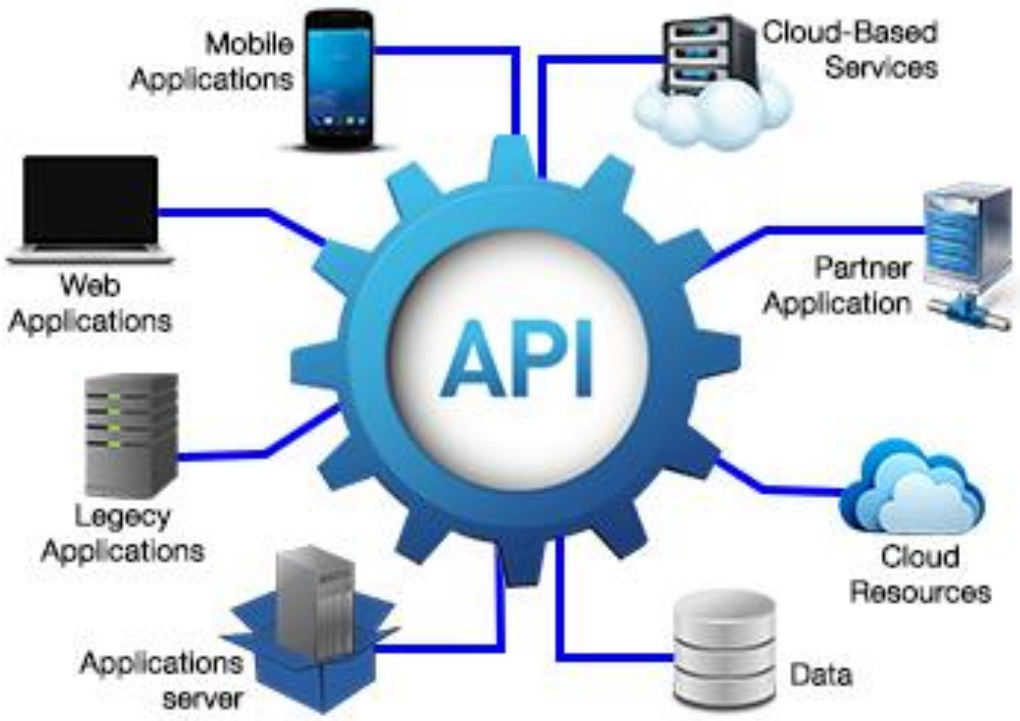


https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/

What is a Web API?

- Is an API over the web which can be accessed using HTTP protocol
- Is a concept and not a technology
- Usually exposes back-end services and therefore does not provide user interfaces
- Request/Response messages are defined in JSON or XML
- The third-party software accesses the Web API from exposed endpoints

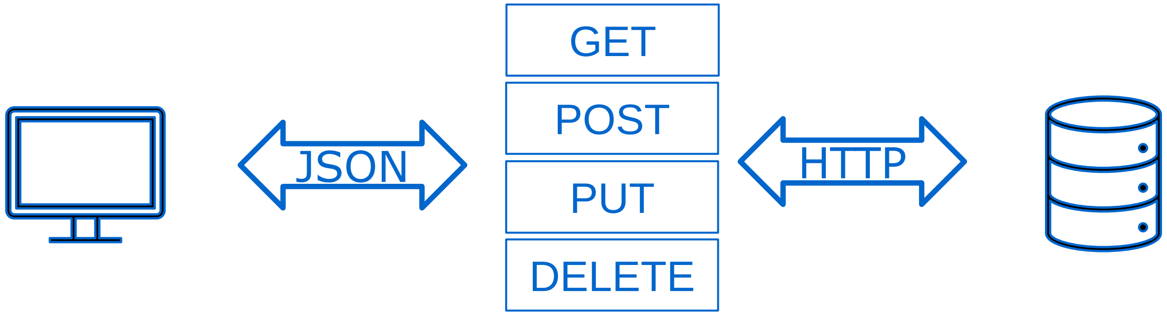
What is a Web API?



https://www.intergate.net.br/blog/a-estrutura-de-uma-api-rest/

What is a RESTful Web API?

- Architectural style for an application program interface (API) that uses HTTP requests to access and use data
- The data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.



UPskill

Digital Skills & Jobs

Development Environment

Desenvolvimento de Aplicações Web

P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

Required Environment

- Visual Studio Community 2022 ⤓

  "Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code."

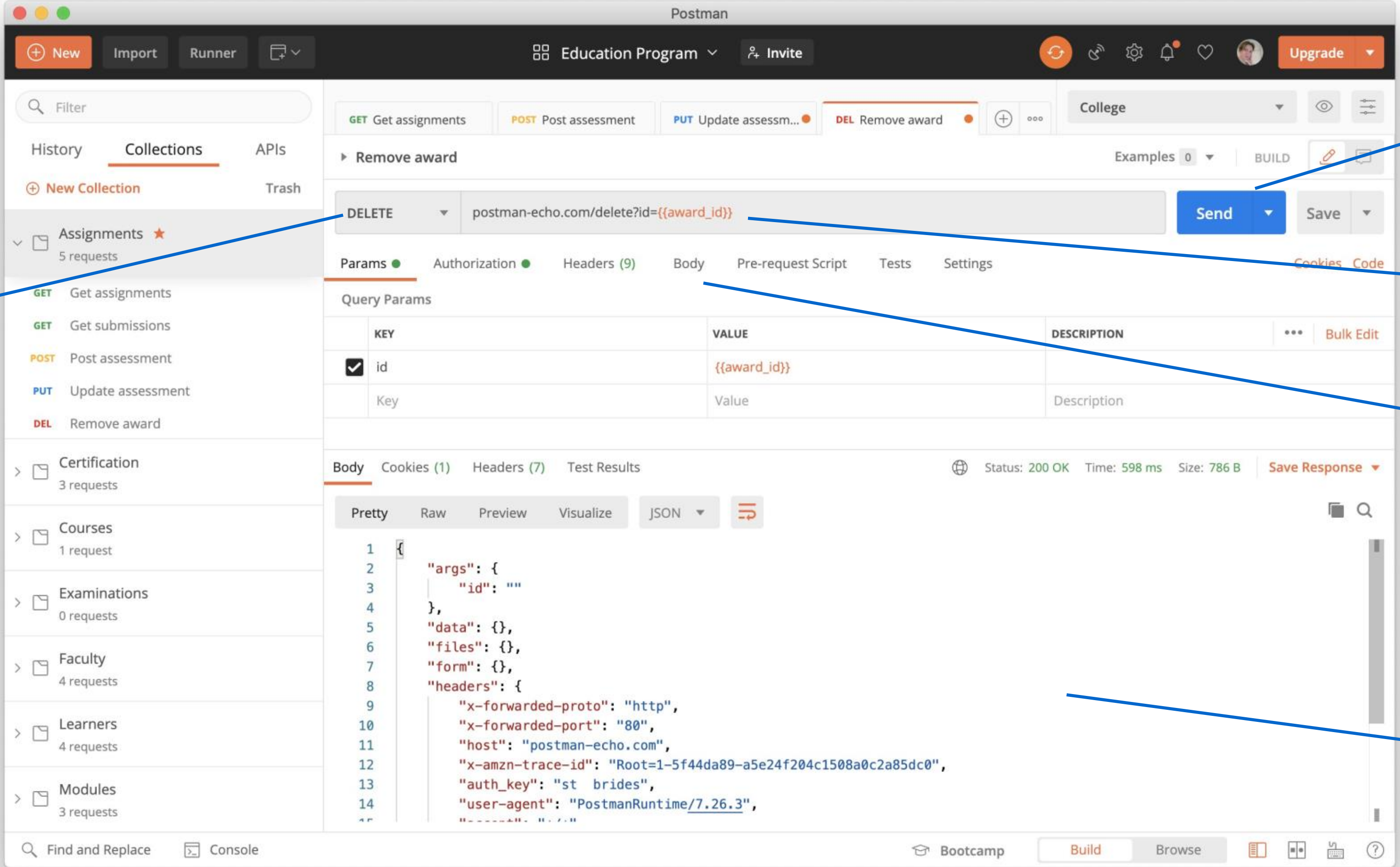
- SQLite ⤓

  "SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day"

- Postman ⤓

  "Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs—faster."

Postman



Postman

New | Import | Runner | Education Program | Invite | Upgrade

Filter

History | Collections | APIs

New Collection | Trash

- Assignments ★ — 5 requests
  - GET Get assignments
  - GET Get submissions
  - POST Post assessment
  - PUT Update assessment
  - DEL Remove award
- Certification — 3 requests
- Courses — 1 request
- Examinations — 0 requests
- Faculty — 4 requests
- Learners — 4 requests
- Modules — 3 requests

GET Get assignments | POST Post assessment | PUT Update assessm... | DEL Remove award

College

Remove award

Examples 0 | BUILD

DELETE | postman-echo.com/delete?id={{award_id}} | Send | Save

Params ● | Authorization ● | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies | Code

Query Params

	KEY	VALUE	DESCRIPTION
☑	id	{{award_id}}	
	Key	Value	Description

Body | Cookies (1) | Headers (7) | Test Results

Status: 200 OK | Time: 598 ms | Size: 786 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
{
    "args": {
        "id": ""
    },
    "data": {},
    "files": {},
    "form": {},
    "headers": {
        "x-forwarded-proto": "http",
        "x-forwarded-port": "80",
        "host": "postman-echo.com",
        "x-amzn-trace-id": "Root=1-5f44da89-a5e24f204c1508a0c2a85dc0",
        "auth_key": "st  brides",
        "user-agent": "PostmanRuntime/7.26.3",
```

Find and Replace | Console | Bootcamp | Build | Browse

Send Request

Resource

HTTP verb

Input body.
Choose JSON

Response

api/category/

Exercise

- https://official-joke-api.appspot.com/random_joke
- https://vodsystem.onrender.com/doc/
- http://193.136.62.24/swagger-ui.html

UPskill

Digital Skills & Jobs

ASP.NET Web API

Programação Orientada a Objetos

P.PORTO

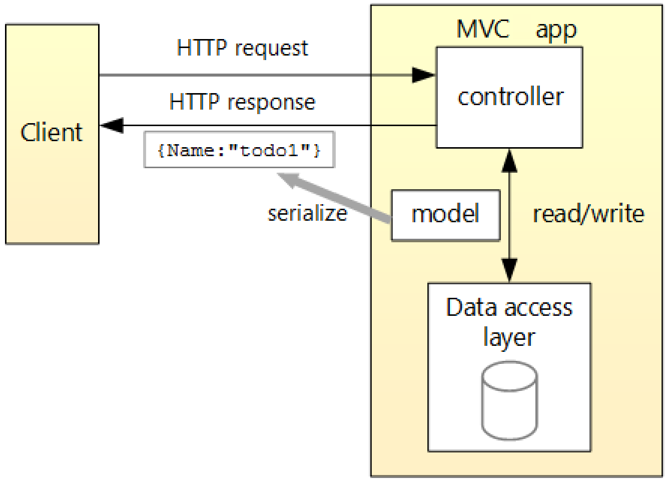
apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

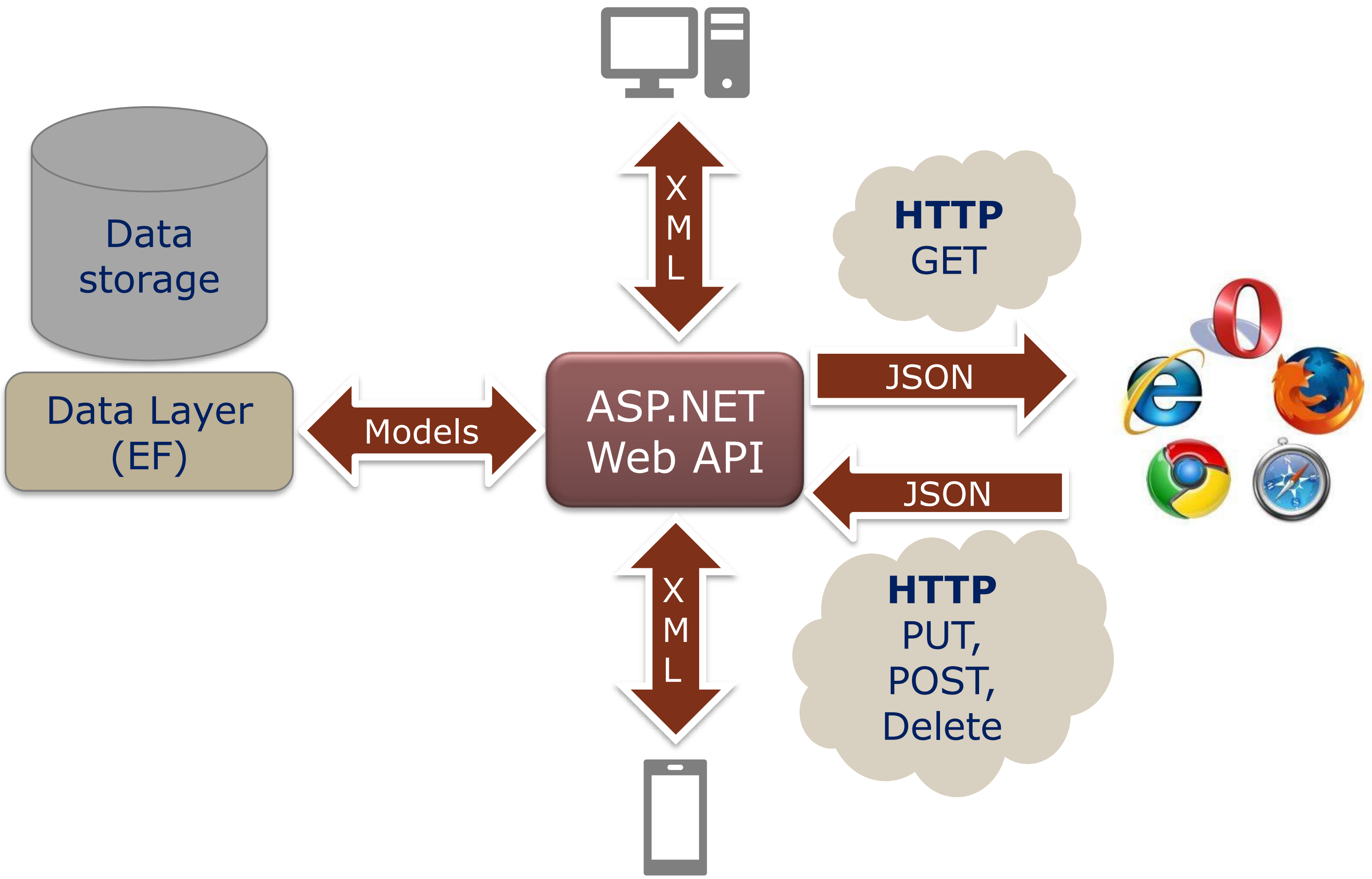
CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

What is ASP.NET Web API?

- Framework that makes it easy to build HTTP services for browsers and mobile devices
- Platform for building RESTful applications on the .NET Framework core using ASP.NET stack



Scenario



Data storage
Data Layer (EF)
Models
ASP.NET Web API
XML
XML
HTTP GET
JSON
JSON
HTTP PUT, POST, Delete

ASP.NET Web API

- ASP.NET Web API is a Framework for building HTTP Services on top of the .Net Framework.
- Routing in MVC:
  - `{controller}/{action}/{id}`

In MVC routing we have the ***action***

- Routing in ASP.NET Web API :
  - `api/{controller}/{id}`
- In Web API the action is determined by the HTTP verb (post, get, put, delete, ...)

ControllerBase class

- A Web API consists of one or more controller classes that derive from `ControllerBase`
  - The Web API project template provides a starter controller
- Don’t create a Web API controller derived from the `Controller` class
  - Controller derives from `ControllerBase` and adds support for views
  - It’s for handling web pages and not web API requests

ControllerBase class

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using TodoApi.Models;

namespace TodoApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TodoItemsController : ControllerBase
    {
        private readonly TodoContext _context;

        public TodoItemsController(TodoContext context)
        {
            _context = context;

            if (_context.TodoItems.Count() == 0)
            {
                _context.TodoItems.Add(new TodoItem { Name = "Item1" });
                _context.SaveChanges();
            }
        }
    }
}
```

Decorates the class with the **[ApiController]** attribute. This attribute indicates that the controller responds to web API requests.

Inject the database context (**TodoContext**) into the controller. The database context is used in each of the CRUD methods in the controller.

Adds an item to the database if the database is empty. This code is in the constructor, so it runs every time there's a new HTTP request. If you delete all items, the constructor creates Item1 again the next time an API method is called.

Routing

- Web API selects actions based on HTTP methods
- By default, the Web API looks for a match between an HTTP method and the beginning of the controller action method.
  - method `PutCustomers` → HTTP request `HTTP PUT`
- This convention may be overpass by using the annotations:

▪ `[HttpDelete]`	▪ `[HttpPatch]`	▪ `[HttpOptions]`
▪ `[HttpGet]`	▪ `[HttpPost]`	
▪ `[HttpHead]`	▪ `[HttpPut]`	

Routing attributes

- Allows to differentiate routings
  - [Route ("<expressão>")]
- Examples (assuming *book* controller)::
  - [Route("{id:int}")]
    - ○ *api/book/5*
  - [Route("title/{nome}")]
    - ○ api/book/title/Game Of Thrones
  - [Route("{pubdate:datetime:regex(^\\d{4}-\\d{2}-\\d{2}$)}")]
    - ○ api/book/2018-06-24

Routing annotations

```
[Route("api/books")]
public class BooksController : ControllerBase
{
    [HttpGet]
    public Task<ActionResult<IEnumerable<Book>>> GetBooks()
    {
       return await _context.Books.ToListAsync();
    }

    [HttpGet("{id}")]
    public Task<ActionResult<Book>> GetBook(long id)
    {
       var book = await _context.Books.FindAsync(id);

       if (book == null)
       {
          return NotFound();
       }

       return book;
    }

    [HttpPost]
    public HttpResponseMessage CreateBook(Book book) { ... }
}
```

These methods implements:

- GET /api/books
  - *http://localhost:3000/api/books*
- GET /api/books/{id}
  - *http://localhost:3000/api/books/1*
- POST /api/books
  - *http://localhost:3000/api/books*

The following HTTP response is produced by the call to *GetBooks*:

```
[
   {
      "id" : 1,
      "title" : "Web API Bible",
      "author" : "John Smith"
   }
]
```

Routing – Route Prefix

- In the same controller it is usual for routing to start in the same route prefix

```
public class BooksController : ControllerBase
{
    [Route("api/books")]
    public Task<ActionResult<IEnumerable<Book>>> GetBooks()
    { ... }

    [Route("api/books/{id}")]
    public Task<ActionResult<Book>> GetBook(long id)
    { ... }

    [Route("api/books")]
    [HttpPost]
    public HttpResponseMessage CreateBook(Book book)
    { ... }
}
```

Routing – Route Prefix

- In the same controller it is usual for routing to start in the same route prefix

```
[RoutePrefix("api/books")]
public class BooksController : ControllerBase
{
    [Route("")]
    public Task<ActionResult<IEnumerable<Book>>> GetBooks()
    { ... }

    [Route("{id}")]
    public Task<ActionResult<Book>> GetBook(long id)
    { ... }

    [Route("")]
    [HttpPost]
    public HttpResponseMessage CreateBook(Book book)
    { ... }
}
```

Routing – Route Prefix

- Use a tilde (~) on the method attribute to override the route prefix:

```
[RoutePrefix("api/books")]
public class BooksController : ControllerBase
{

 (…)

    // GET /api/authors/1/book
    [Route("~/api/authors/{authorId:int}/books")]
    public Task<ActionResult<IEnumerable<Book>>> GetBooksByAuthor(long id)
    { ... }

 (…)

}
```

Routing – QueryString

- How to execute the request:

```
GET : api/categoria/?descricao=armarios de garagem
```

- In other words
  - To execute a route that responds to the GET method in the `categoria` controller.
- Problem:
  - QueryString (in particular the symbol '?') cannot be specified within a route

Routing – QueryString

- Obvious solution

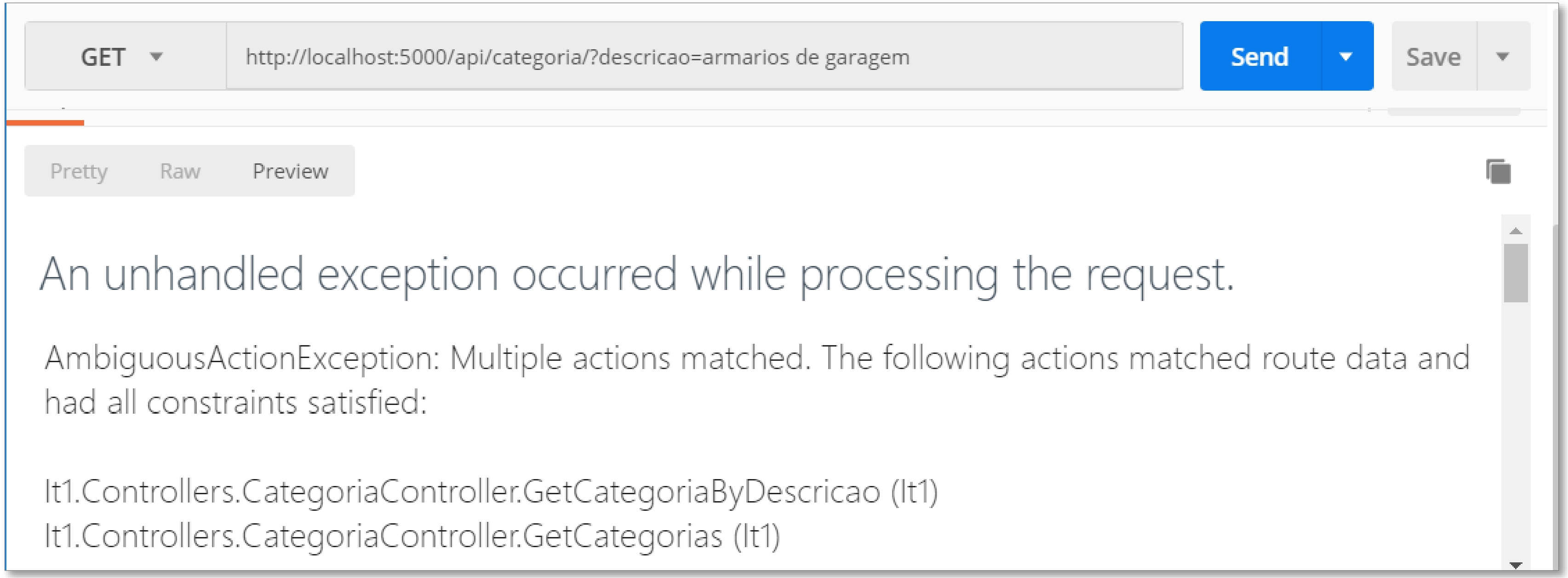
```
api/categoria/?descricao=armarios de garagem
```

```
// GET api/category/?descricao={nome}
[HttpGet]
public IQueryable<Categoria> GetCategoriaByDescricao(
                                   [FromQuery(Name = "descricao")] string desc)
{
    IQueryable<Categoria> categoria = _context.Categorias.
        Where(c => c.Descricao.Equals(desc, StringComparison.OrdinalIgnoreCase));

    return categoria;
}
```

Routing – QueryString

- However, there are **two methods** in the controller that respond to GET, which may cause ambiguity in the response.



http://localhost:5000/api/categoria/?descricao=armarios de garagem

An unhandled exception occurred while processing the request.

AmbiguousActionException: Multiple actions matched. The following actions matched route data and had all constraints satisfied:

It1.Controllers.CategoriaController.GetCategoriaByDescricao (It1)
It1.Controllers.CategoriaController.GetCategorias (It1)

Routing – QueryString

- The solution is to remove ambiguity:

```
api/categoria/?descricao=armarios de garagem
```

```
(…)

// GET api/Categoria
[HttpGet]
public IEnumerable<Categoria> GetCategorias()
{
     if(Request.Query.ContainsKey("descricao"))
               return GetCategoriaByDescricao(Request.Query["descricao"]);

     //conteúdo restante do GetCategorias
     (…)
}

(…)
```

Routing – QueryString

```
api/categoria/?descricao=armarios de garagem
```

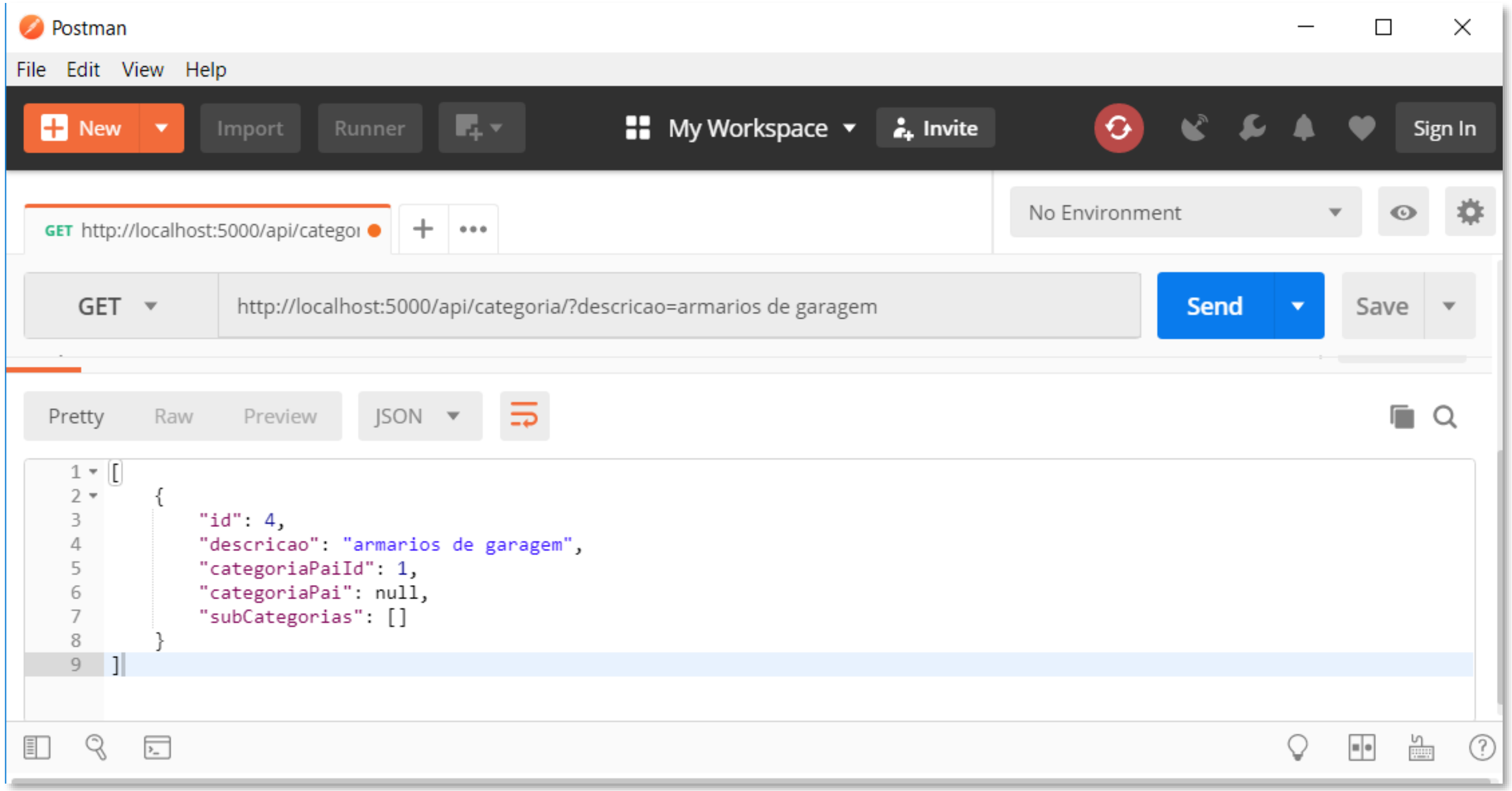
```
(…)

public IQueryable<Categoria> GetCategoriaByDescricao(string desc)
{
    IQueryable<Categoria> categoria = _context.Categorias.
        Where(c => c.Descricao.Equals(desc, StringComparison.OrdinalIgnoreCase));

    return categoria;
}

(…)
```

Routing – QueryString



GET http://localhost:5000/api/categoria/?descricao=armarios de garagem

```
[
    {
        "id": 4,
        "descricao": "armarios de garagem",
        "categoriaPaiId": 1,
        "categoriaPai": null,
        "subCategorias": []
    }
]
```

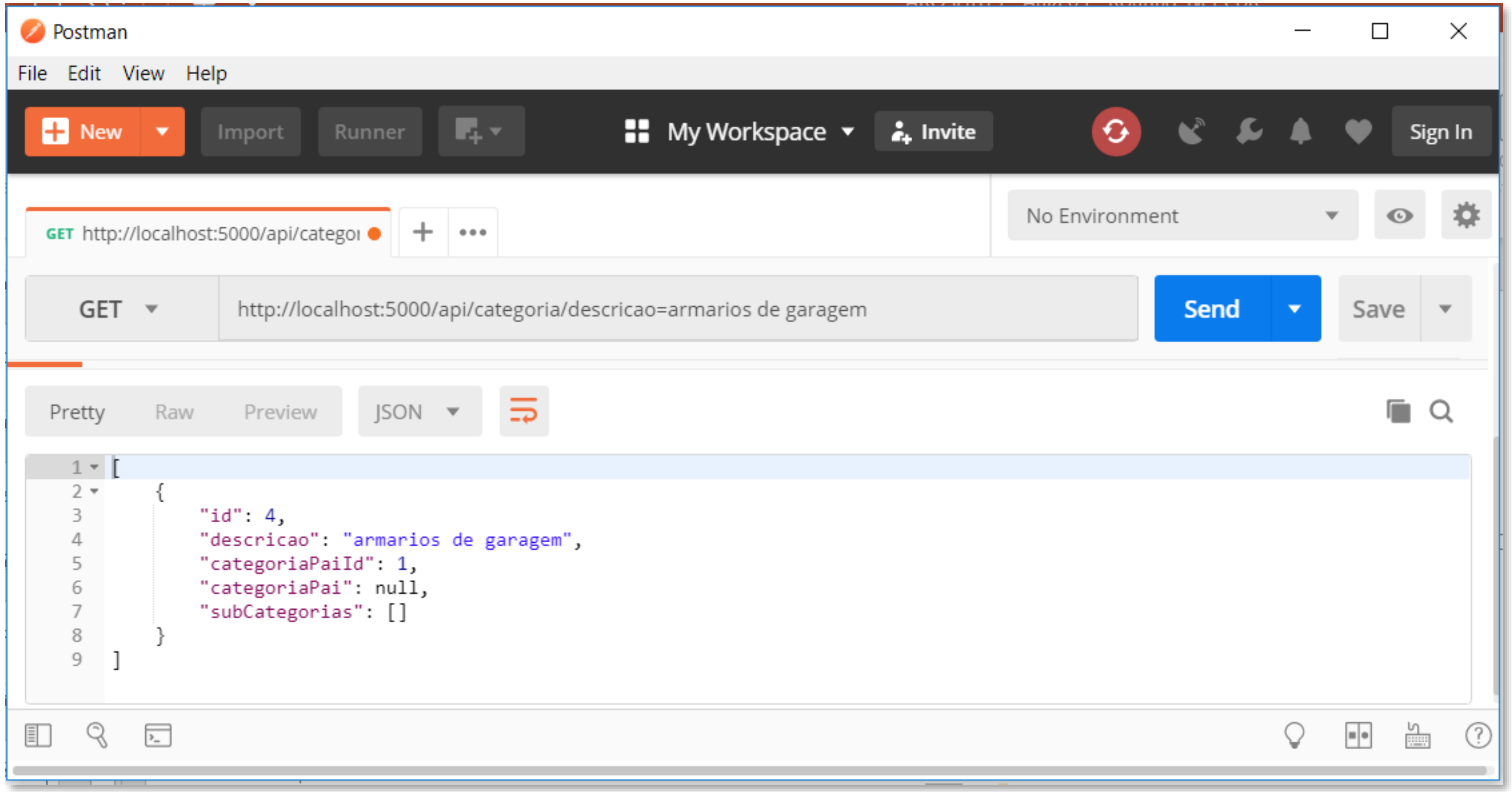
Routing – dropping the "?"

```
api/categoria/descricao=armarios de garagem
```

```
// GET api/Categoria/descricao={nome}
[HttpGet("descricao={desc}")]
public IQueryable<Categoria> GetCategoriaByDescricao([FromRoute] string desc)
{
    IQueryable<Categoria> categoria = _context.Categorias.
        Where(c => c.Descricao.Equals(desc, StringComparison.OrdinalIgnoreCase));

    return categoria;
}
```

Routing – dropping the "?"



GET http://localhost:5000/api/categoria/descricao=armarios de garagem

```
[
    {
        "id": 4,
        "descricao": "armarios de garagem",
        "categoriaPaiId": 1,
        "categoriaPai": null,
        "subCategorias": []
    }
]
```

UPskill

Digital Skills & Jobs

DTO

Programação Orientada a Objetos

P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

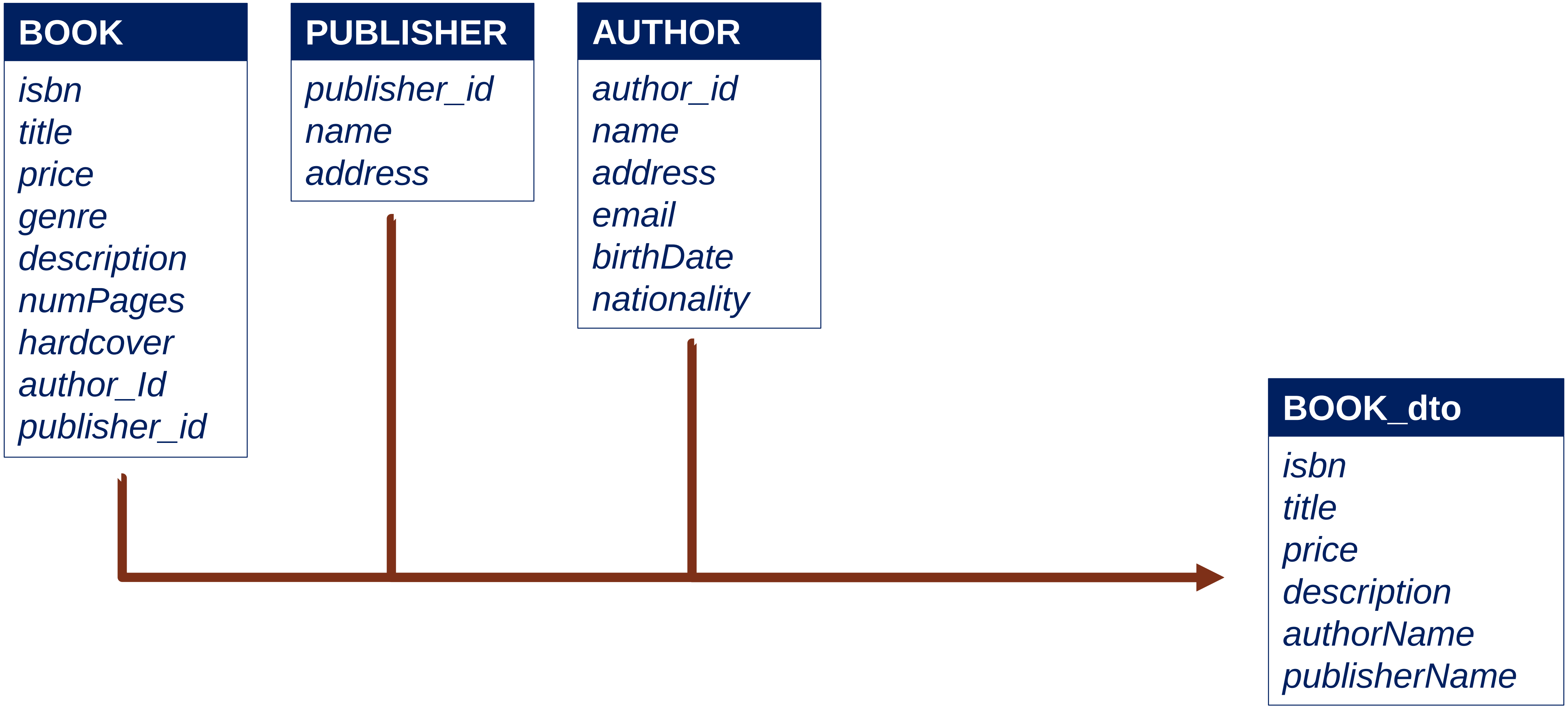
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITÉCNICOS

What is a DTO?

- Acronym for **D**ata **T**ransfer **O**bject
- DTO is an object that is designed to carry data between processes
- DTO doesn't have any business rules

What is a DTO?



BOOK

isbn
title
price
genre
description
numPages
hardcover
author_Id
publisher_id

PUBLISHER

publisher_id
name
address

AUTHOR

author_id
name
address
email
birthDate
nationality

BOOK_dto

isbn
title
price
description
authorName
publisherName

DTO Example

Class Book.cs

```
namespace BooksAPI.Models {
   public class Book {
      public int BookId { get; set; }
      [Required]
      public string Title { get; set; }
      public decimal Price { get; set; }
      public string Genre { get; set; }
      public string Description { get; set; }
      public int AuthorId { get; set; }
      [ForeignKey("AuthorId")]
      public Author Author { get; set; }
   }
}
```

Class Author.cs

```
namespace BooksAPI.Models
{
   public class Author {
      public int AuthorId { get; set; }
      [Required]
      public string Name { get; set; }
   }
}
```

DTO Example

Class Book.cs

```
namespace BooksAPI.Models {
   public class Book {
      public int BookId { get; set; }
      [Required]
      public string Title { get; set; }
      public decimal Price { get; set; }
      public string Genre { get; set; }
      public string Description { get; set; }
      public int AuthorId { get; set; }
      [ForeignKey("AuthorId")]
      public Author Author { get; set; }
   }
}
```

Request to /api/books/1

```
{
       "BookId": 1,
       "Title": "The Design of Web APIs",
       "Genre": "web development",
       "Description": "Web APIs are
          everywhere, giving developers an
          efficient way to interact with
          applications and services.",
       "Price": 37.15,
       "AuthorId": 1,
       "Author": null
}
```

DTO Example

Class Book.cs

```
namespace BooksAPI.Models {
   public class Book {
      public int BookId { get; set; }
      [Required]
      public string Title { get; set; }
      public decimal Price { get; set; }
      public string Genre { get; set; }
      public string Description { get; set; }
      public int AuthorId { get; set; }
      [ForeignKey("AuthorId")]
      public Author Author { get; set; }
   }
}
```

Class BookDTO.cs

```
namespace BooksAPI.DTOs {
   public class BookDTO {
      public string Title { get; set; }
      public decimal Price { get; set; }
      public string Genre { get; set; }
   }
}
```

Add a New Folder, Name the folder "DTOs”
Add a class BookDTO.cs

DTO Example

Class Book.cs

```
namespace BooksAPI.Models {
   public class Book {
      public int BookId { get; set; }
      [Required]
      public string Title { get; set; }
      public decimal Price { get; set; }
      public string Genre { get; set; }
      public string Description { get; set; }
      public int AuthorId { get; set; }
      [ForeignKey("AuthorId")]
      public Author Author { get; set; }
   }
}
```

Request to /api/books/1

```
{
       "Title": "The Design of Web APIs",
       "Price": 37.15,
       "Genre": "web development"
}
```

UPskill

Digital Skills & Jobs

Entity Framework

Programação Orientada a Objetos

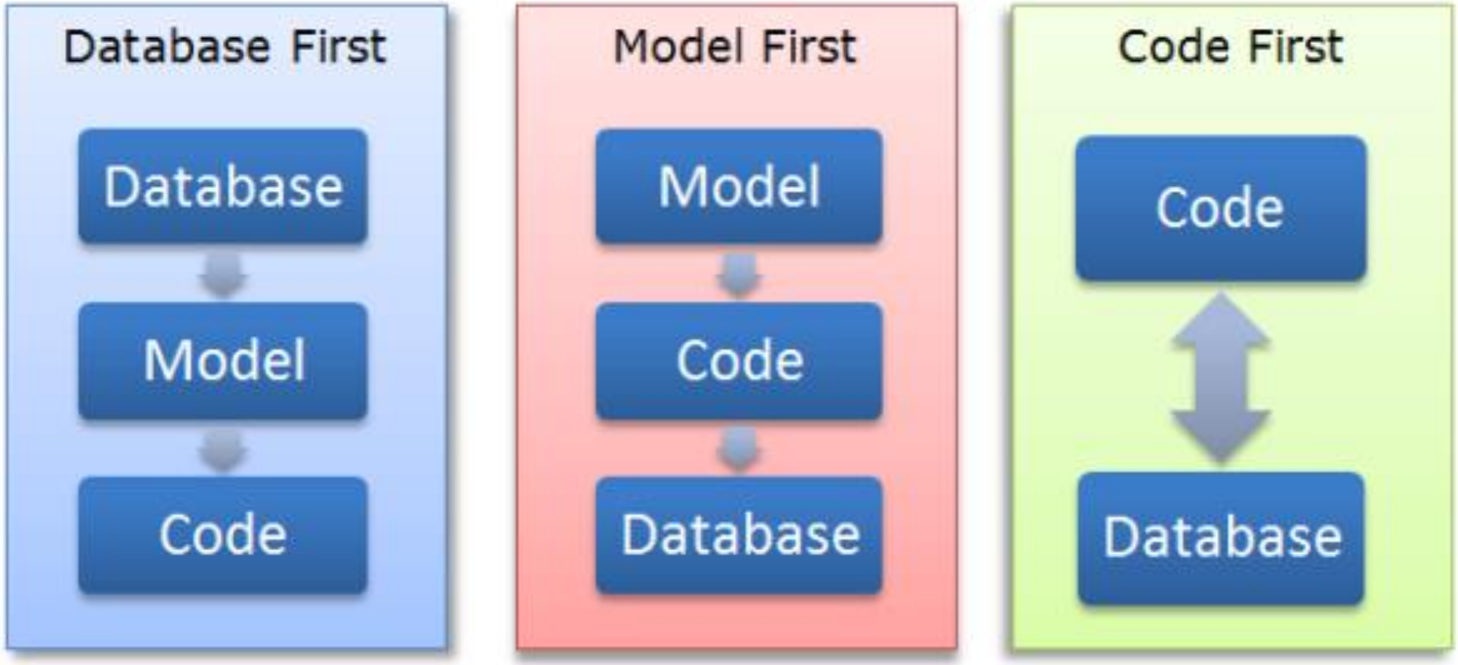
P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

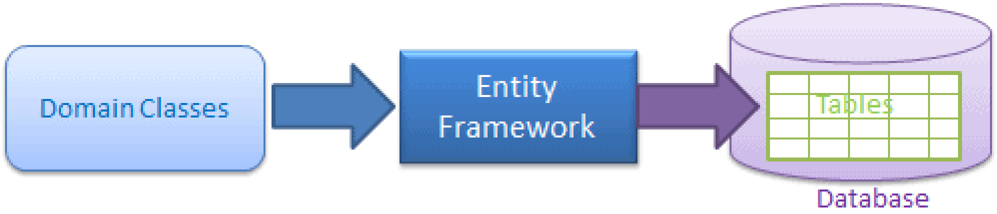
CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS

Several Approaches

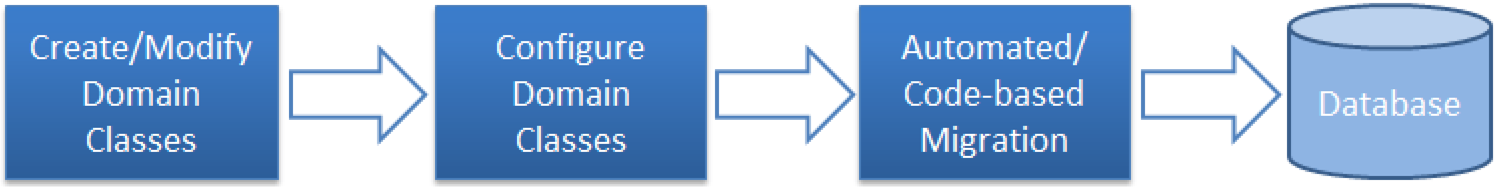


What is Code-First?

- In the Code-First approach, you focus on the application's domain and create classes for the domain entities. The EF will create the database based on domain classes and configurations.



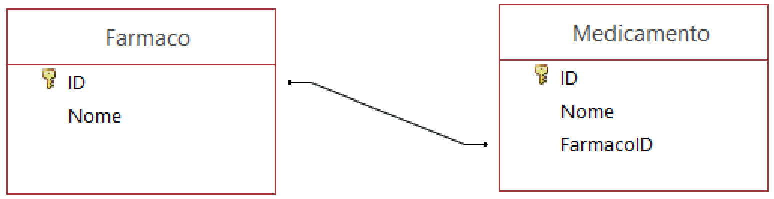
- The code-first development workflow is:



Querying in Entity Framework

- Eager Loading
  - Process whereby a query for one type of entity also loads related entities as part of the query, so that we don't need to execute a separate query for related entities.
  - Eager loading is achieved using the `Include()` method.
- Lazy Loading
  - Lazy loading is delaying the loading of related data, until you specifically request for it. It is the opposite of eager loading.
  - Default behavior from EF Core 2.1
- Explicit Loading
  - Load related entities in an entity graph explicitly

Querying in Entity Framework



api/Medicamentos

```
{
    "id": 1,
    "nome": "Brufen",
    "farmacoId": 1,
    "farmaco": null
}
```

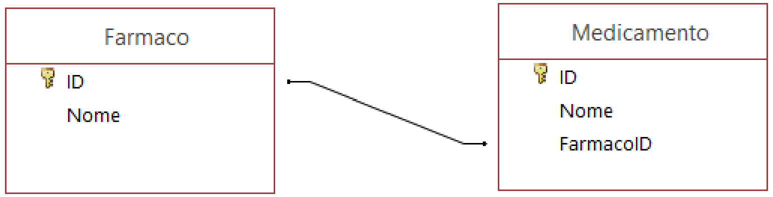
Eager Loading

- EF obtains related entities as part of the initial database query
- Use System.Data.Entity.Include in the controller action method

```
(…)
    // GET: api/Medicamentos
    [HttpGet]
    public IEnumerable<Medicamento> GetMedicamento()
    {
        return _context.Medicamento.Include(f => f.Farmaco);
    }
(…)
```

lambda expression

Eager Loading



api/Medicamentos

```
{
    "id": 1,
    "nome": "Brufen",
    "farmacoId": 1,
    "farmaco": {
         "id": 1,
         "nome": "ibuprofeno"
    }
}
```

Lazy Loading

- EF automatically obtains a related entity when the navigation property for that entity is referenced.
- Classify the navigation property as `virtual` in the entity definition

```
public class Medicamento
{
        public int id { get; set; }
        public string Nome { get; set; }
        public int FarmacoId { get; set; }

        virtual public Farmaco Farmaco { get; set; }
}
```

Lazy Loading

- Lazy Loading requires several interactions with BD as EF will make a query for each entity that has to recover
- So, this method **IS NOT SUITABLE** for entity serialization situations (as is the case with the Web APIs)
- One solution to this problem is the use of DTO’s

Explicit Loading

- Similar to Lazy Loading with the particularity that related entities are explicitly obtained in code (they are not loaded by simple access to a navigation property).

- Provides greater control over the loading of related information but requires greater coding effort

UPskill

Digital Skills & Jobs

ASP.NET Web API

Programação Orientada a Objetos

P.PORTO

apdc ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITECNICOS