Getting Data from APIs

Nicolás Dussaillant

Data Science and Education Association (DSEA)

Teachers College, Columbia University

Why this workshop on this topic?

In the last years many applications, services, datasets, web servers are offering public (or available by request) access to their data through APIs. Since most of us here work with data, this can be a good tool to considering when downloading data.

Material: tinyurl.com/workshopapi

Objectives of the session

- To broadly understand how APIs work
- To recognize important parts of the process to use one to get data
- To understand (and replicate) a real example of using an API

Agenda

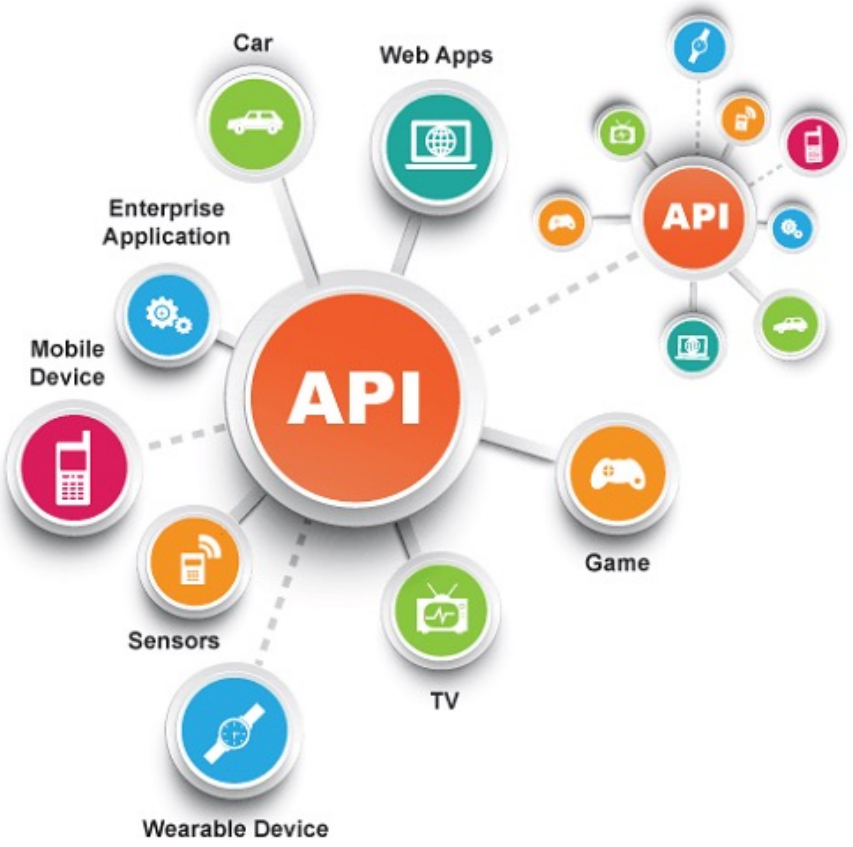
- What APIs are and their different potential usages
- Relevant concepts to understand the usage of APIs and how they usually work
- Reasons to use and API and some good scenarios where they could be beneficial to use, and guidelines to start using one
- Brief examples of checking and using an API (using Python)

Some concepts

- API
- HTTP
- URL (URI)
- JSON
- XML
- CSV

Quick Explanations

API: Application Programming Interface



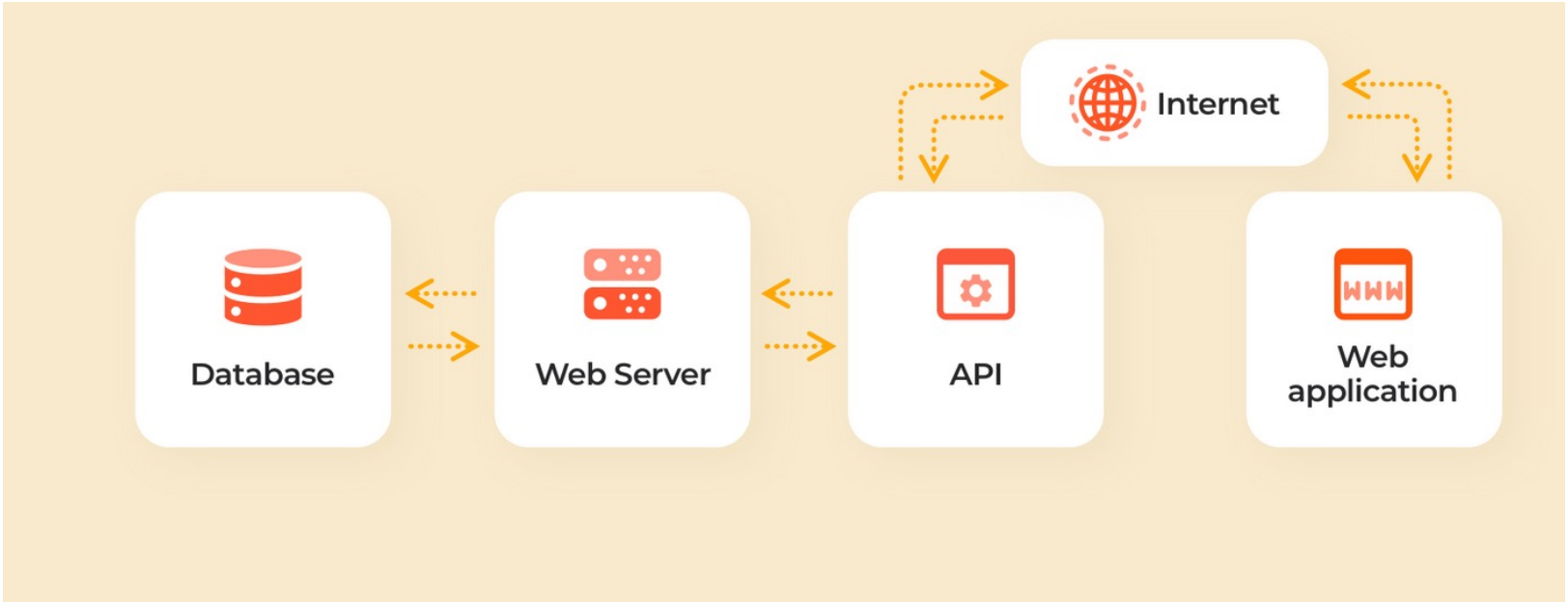


Image: https://www.cleveroad.com/blog/what-is-an-api

HTTP: Hypertext Transfer Protocol

- Internet communication protocol in the application layer
- Designed to be a **request-response** protocol in a **client-server** model
- Different types of requests. In this context we typically use GET and sometimes POST.
- Data goes in packages with different components, the most important difference between components are **headers** and **content**
- HTTPS: HTTP but adding security

HTTPS: HTTP Secure

- HTTP with SSL/TLS encryption
- Basically, the communication cannot be spied by intermediate network points between client and server

URI: Uniform Resource Identifier
URL: Uniform Resource Locator

- A URL is a type of URI, it's used to identify the location where we are requesting a response.

Basic structure example:

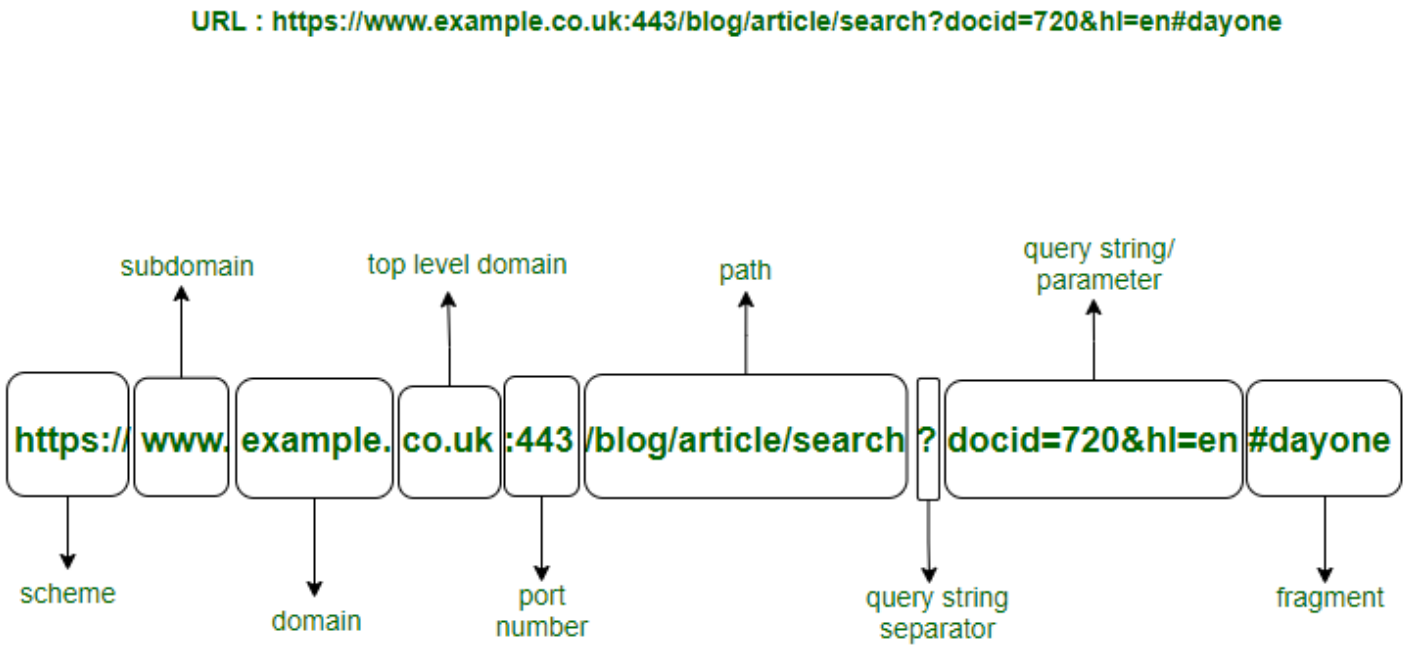


Image:
https://www.geeksforgeeks.org/components-of-a-url/

URL Encoding

- Not every character is allowed in a URL and some are used for other things
- Some libraries from the programming language can help you with this
- You can check the allowed characters and how to escape or encode the ones that are not allowed:

  https://developers.google.com/maps/url-encoding

JSON: JavaScript Object Notation

- Data format
- Extension for file storage: .json
- Example:

```
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
            "title": "S",
                        "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                    },
                                        "GlossSee": "markup"
                }
            }
        }
    }
}
```

From: https://json.org/example.html

XML: Extensible Markup Language

- Data format
- Extension for file storage: .xml
- Example:

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

From: https://json.org/example.html

CSV: Comma Separated Values

- Data format
- Extension for file storage: .csv
- Typical separator is ”,” but some use “;” instead
- Example:

id, name, age, nationality
1902, “Nicolas Dussaillant”, 30, “Chilean”
872, “Roger Federer”, 40, “Swiss”

Some reasons to use an API to retrieve data
(for most of our contexts here)

- Can filter, paginate, and combine data before downloading it
- Data can be automatically updated
- Data can be read directly from the code or statistical program (if it supports it)
- Sharing the code does not require sending the data if the API is still available
- Comparatively with database servers, APIs can give more flexibilities, restrictions, and easier protocols

Checking an API before using it

Some guidelines before starting using an API to get data

- Check the availability of the API
- Look into the details of the API in the dedicated website
- Check the available datasets in the API and get data dictionaries and meta data
- Check if authentication is required and how it’s done
- Read the documentation of the API protocol or find libraries that could help you handle it
- Check how the results will be delivered and how you can parse and use them
- Check the tools given by the API (filters, formats, etc.)
- Check if there are restrictions to the usage (max number of requests)

Next parts:

- Check the documentation of an API

- Run some code and try an API

https://github.com/njdussai/api_workshop

tinyurl.com/workshopapi

CollegeScoreboard API

Dataset from the US Government with information about higher education institutions.

https://collegescorecard.ed.gov/

NYC Open Data

Datasets from New York City. Several different topics, some more updated than others.

https://opendata.cityofnewyork.us/

Thanks!

njd2137@tc.columbia.edu