

# Programmable Web Project

## Part 1: Introduction

Spring 2019

## The World Wide Web

## Technologies for the World Wide Web

- Backend: Business logic + data storage (databases)
  - Transport protocol: HTTP
  - Data serialization languages
  - Clients
- **Services and APIs**
  - **Programmable Web**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



1

2

## The World Wide Web

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



2

3

## What is the World Wide Web?



[https://www.youtube.com/watch?v=OM6XIIcm\\_qo&start=18&end=190&autoplay=1](https://www.youtube.com/watch?v=OM6XIIcm_qo&start=18&end=190&autoplay=1)

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



3

4

## What is the World Wide Web?

### Goal: Distribute data

- Human consumption (H2M)
  - Hypertext
- Uniform API and technologies
- Single client (Web browser)

Iván Sánchez Milara

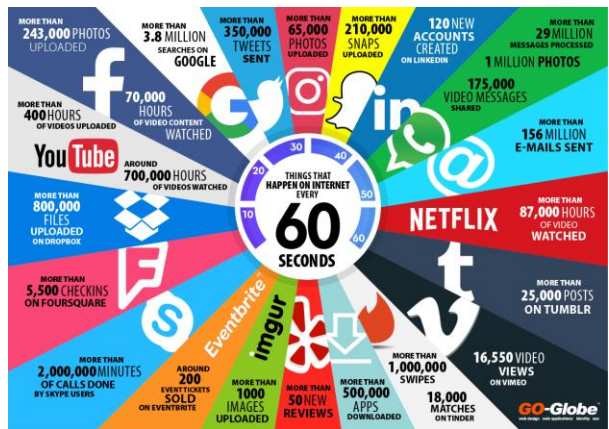
Programmable Web Project. Spring 2019.



4

5

World Wide Web success. Scalability



Web is distributed      Web is massively decoupled      Web is dynamic

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



5

6

How the WWW works?

<http://www.youtypeitwepostit.com/>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



6

7

## TECHNOLOGIES FOR THE WWW

- Backend: Business logic + data storage (databases)
- Transport protocol: HTTP
- Data serialization languages
- Clients

Iván Sánchez Milara

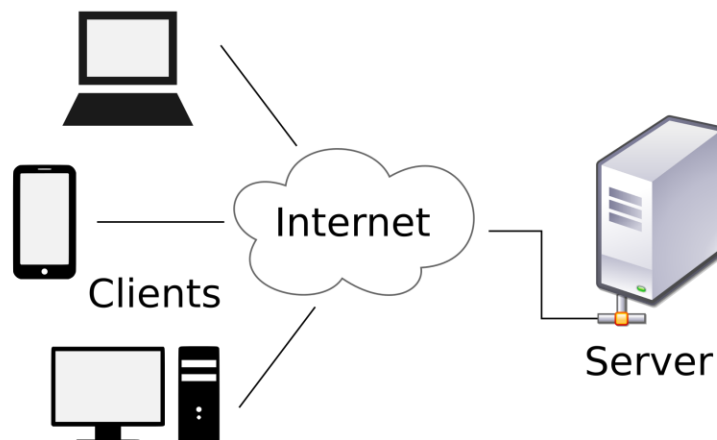
Programmable Web Project. Spring 2019.



7

8

## Client server model



© David Vignoni LGPL license  
[https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model#/file:Client-server-model.svg](https://en.wikipedia.org/wiki/Client%E2%80%93server_model#/file:Client-server-model.svg)

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



8

9

## BACKEND

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



9

10

## Backend

- Stores application data persistently
  - DATABASE
- Defines how to process request from the client and process the data according to the requests coming from the client
  - BUSINESS LOGIC
- Expose the data using a defined API

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



10

11

# DATABASES

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



11

12

## Definition

- Databases emerged to solve challenges of storing and managing huge amounts of data
- A database:
  - is a data structure
  - stores organized information
  - can be easily accessed, managed and updated
- DBMS (Database Managing System) is the software that allows creating, managing and storing database structures.
  - Responsible for data integrity, recovery and access
  - Provides a way for extract or modify the data
- There are different ways to model the data in the database
  - Lately divided into relational models and non-relational models

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



12

14

## Relational – Non-relational

- **Relational:**

- Database model developed by E.F. Codd in 1970
  - Codd, Edgar F (June 1970). "[A Relational Model of Data for Large Shared Data Banks](#)"
- Data is represented in terms of tuples (rows), grouped into relations (tables) that can be linked with each other.
- Developed almost in parallel with SQL language

- **Non-Relational:**

- Sometimes miscalled Non SQL databases
- Umbrella that gathers different databases that are not relational.
- Data is not organized in related tables.
  - Some store objects, some store key-value pairs, some store documents
- More flexible and scalable

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



14

15

## RDBMS Concepts

- **ORM**

- To access a relational database from an object oriented context (PHP, python, Java...)
  - interface translating relational logic to objects logic is needed.
  - Such interface is called **Object-relational mapping (ORM, O/RM, and O/R mapping)**.

- **CRUD**

- Databases are persistent data
- There are four basic functions to manage persistent data:
  - Create
  - Read
  - Update
  - Delete

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



15

### SQL vs NoSQL vs NewSQL

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No



### Examples - Relational

- Relational databases are still the most commonly used.
- Relational databases are mainly composed by tables.
- A table is formed by zero (empty) or more rows.
- A row consists of one or more fields
  - Each has a certain datatype. (columns)

FirstName	Surname	PersonalId
John	Smith	3321
Jack	Johnson	4352
Mary	Smith	9807

- Some examples are: PostgreSQL, MySQL, SQLite





18

## Examples – Non-relational

### – MongoDB

- Scalable, open source database
- JSON based data store: BSON
- Document-oriented database
  - Database formed by Collections of Documents
- Example of MongoDB document:

```
{
  name: "jim",
  surname: "smith",
  grade: 3
}
```

- Example of MongoDB query:

```
db.students.find({grade:{ $gt:3}});
```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



18

23

## TRANSPORT PROTOCOL: HTTP

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



23

24

## HTTP (I)

### • The Hypertext Transfer Protocol (HTTP):

*"an **application-level protocol** for distributed, collaborative, hypermedia information systems"*

RFC 2616 (<http://www.faqs.org/rfcs/rfc2616.html>)

- HTTP communication usually takes place over TCP/IP connections.
- Most used **application** protocol in the World Wide Web.
- Also used as a transport protocol for other application protocols, such as SOAP, XML-RPC ...
- HTTP allows bidirectional transfer of resources representations between client and server.
  - **Resource**: network data object identified by a URI

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



24

26

## HTTP Request parts

### • HTTP request example to <http://www.cse.oulu.fi>

The **HTTP method**. Here, the client (web browser) is trying to GET some information from the server ([www.cse.oulu.fi](http://www.cse.oulu.fi)).

The **path** In this example the path points to the root of the host (just /)

```

REQUEST LINE
GET / HTTP/1.1
Keep-Alive: 300
Connection: keep-alive
Host: www.cse.oulu.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

```

The **request headers** Since the request does not have entity, it only contains general and request specific headers.

The **entity-body** This particular request has no entity body, which means the envelope is empty! This is typical for a GET request, where all the information needed to complete the request is in the path and the headers.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



26

28

HTTP Response parts

- Response Example: `http://www.cse.oulu.fi`

The **HTTP response code**. In this case the GET operation must have succeeded, since the response code is 200 ("OK").

The **response headers**: general, response and entity headers


STATUS LINE

```
HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Date: Wed, 05 Oct 2011 17:26:03 GMT
Server: Apache/2.2.3 (CentOS)
Vary: Cookie, User-Agent, Accept-Language
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="robots" content="index, follow">
<MainPage - Department of Computer Science and Engineering</title>
...
```

The **entity-body**. In this case, the entity body is a HTML document representing a web page.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



28

29


HTTP Methods

Defined in RFC2616

Method	Description
GET	Returns the resource representation
HEAD	Identical to GET except that the server returns only headers information in the response
PUT	Changes the state of the resource Creates a new resource when the URL is known
POST	Create subordinate resources (no URL known beforehand) Appends information to the current resource state
DELETE	Removes a resource from the server

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



29

33

# DATA SERIALIZATION LANGUAGES

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



33

34

## Hypermedia

- Techniques to integrate content in multiple formats (text, image, audio, video...) in a way that all content is connected and accessible to the user.

*"Hypertext [...] the simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions. Machines can follow links when they understand the data format and the relations type"*

Roy Fielding, "[A little REST and Relaxation](http://www.slideshare.net/royfielding/a-little-rest-and-relaxation)"\*

- Hypermedia
  - **Data**
  - **Hypermedia controls.** Indicates what actions could I do next, what are the target resource to perform the action (link) and how can I perform those actions (http method / response).

\* <http://www.slideshare.net/royfielding/a-little-rest-and-relaxation>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



34

35

## Hypermedia (HTML)

```
<a href="http://www.youtypeitwepostit.com/messages/">
    See the latest messages
</a>
```

```

```

```
<form action="http://www.youtypeitwepostit.com/messages" method="post">
    <input type="text" name="message" value="" required="true" />
    <input type="submit" value="Post" />
</form>
```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



35

36

## JSON and XML

- Formats used for representing data that are heavily used to share data among heterogeneous peers
  - Text format (not binary)
  - Language independent
- Although the two of them can be used for M2M and H2M
  - XML is more human readable oriented
  - JSON is more machine readable oriented
- In the Programmable Web, they are mainly used for data exchange, although they may be used also for data storage.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



36

37

## JSON

- JavaScript Object Notation
- Based on a subset of the JavaScript Language
- Built on two structures:
  - A collection of name/value pairs
  - An ordered list of values
- These structures can be mapped to structures in almost any programming language
- Example

```
{ "widget": {
  "debug": "on",
  "window": {
    "title": "Sample Konfabulator Widget",
    "name": "main_window",
    "width": 500,
    "height": 500
  }
}}
```

<http://www.json.org>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



37

38

## XML

- Extensible Markup Language
  - Markup language: system for annotating a document,
- First intended for data publishing
- Markup based in tags:
 

```
<tag>content</tag>
```
- More info
  - Appendix 1: App1\_XML\_Basics
  - <http://www.w3.org/XML/>
- Example

```
<widget>
  <debug>on</debug>
  <window title="Sample Konfabulator Widget">
    <name>main_window</name>
    <width>500</width>
    <height>500</height>
  </window>
</widget>
```

(<http://www.json.org>)

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



38

# Hypermedia: Collection+JSON

Mime type: [application/vnd.collection+json](#)  
Link: <http://amundsen.com/media-types/collection/>

```
{ "collection":  
  {  
    "version" : "1.0",  
    "href" : "http://www.youtypeitweposit.com/api/",  
    "items" : [  
      { "href" : "http://www.youtypeitweposit.com/api/messages/21818525390699506",  
        "data" : [  
          { "name" : "text", "value" : "Test." },  
          { "name" : "date_posted", "value" : "2013-04-22T05:33:58.930Z" }  
        ],  
        "links" : []  
      },  
      { "href" : "http://www.youtypeitweposit.com/api/messages/3689331521745771",  
        "data" : [  
          { "name" : "text", "value" : "Hello." },  
          { "name" : "date_posted", "value" : "2013-04-20T12:55:59.685Z" }  
        ],  
        "links" : []  
      },  
      { "template" : {  
        "data" : [  
          { "prompt" : "Text of message", "name" : "text", "value" : "" }  
        ]  
      }  
    ]  
  }  
}
```

LIST OF HYPERMEDIA FORMATS IN APPENDIX 3: Hypermedia formats

# CLIENTS

44

## Types of clients

- Human driven clients
  - Decisions made by humans. Importance on how to represent information to humans
- Crawlers
  - It starts following all links iteratively from certain web, executing an algorithm for each link followed
  - E.g. Google
- Monitors
  - Checks the state of a resource periodically
  - E.g. RSS aggregator
- Scripts
  - Simulate an human repeating a determined set of actions (eg. Accessing sequentially a list of links).
- Agents
  - Try to emulate humans who are actively engaged with a problem. Looks to representation and take autonomous decisions based on states.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



44

45

## Web browser. An Human Driven client.

- A web browser is the client for ALL websites and web applications.
- **TECHNOLOGIES:**
  - **HTML**-> Markup language which defines the content to be rendered by the browser
  - **CSS**-> Style sheet language used for describing the look and formatting of a document
  - **JAVASCRIPT**-> Scripting language that listen for events triggered by the users, the network or the host system and execute predefined actions.
  - **AJAX**-> A set of techniques based on Javascript which enable asynchronous interaction between a web browser and a server

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



45



49

## SERVICES AND APIS

Iván Sánchez Milara

Programmable Web Project. Spring 2019.

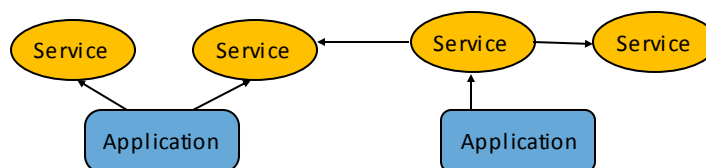


49

50

### Web services (I)

- Web services are logical units with **clearly defined interfaces (API)**:
  - what **functionality** they perform and
  - which **data formats** they accept and produce
- They are **application independent**
  - services can be used by other services and applications.
- Web service can incorporate the functionality of other services (**composite service**)



Iván Sánchez Milara

Programmable Web Project. Spring 2019.



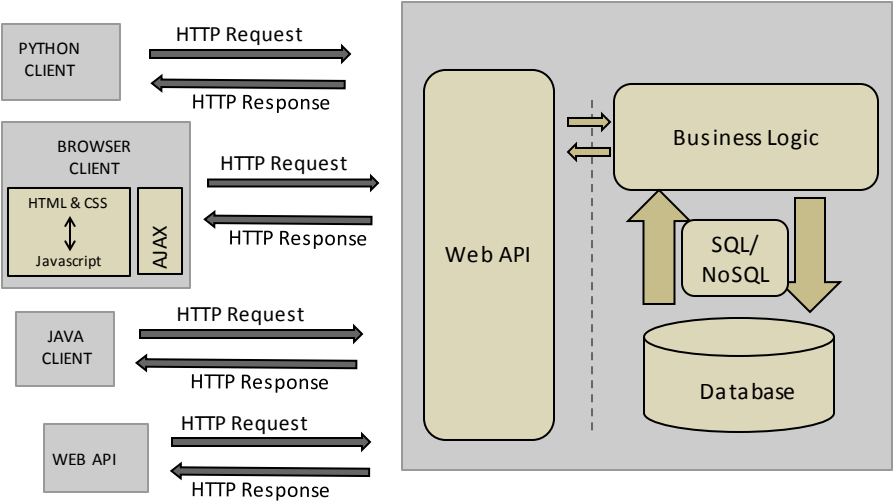
50

# Web services (II)

- Web services are not prepared to human consumption (in contrast to websites).
  - Web services require an architectural style to provide **clear and unambiguous interaction** (clearly defined interfaces), because there's no smart human being on the client end to keep track.



# Web API



53

## Web APIs

- **Application Programming Interfaces**
- Defines how the service functionality is exposed by means of one or more endpoints:
  - Protocol semantics
  - Application semantics
- **Nowadays, web service word is in disuse => We use Web API instead**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



53

54

## Website vs Web API

- **Gist:**
  - Github tool that allows sharing code and applications
  - Website at: <https://gist.github.com/>
  - API at <https://developer.github.com/v3/gists/>
  - Gist clients: <https://gist.github.com/defunkt/370230>
    - For instance, Sublime Text client: <https://github.com/condemil/Gist>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



54

56

## Architectural styles

- RPC
- REST
  - CRUD
  - Hypermedia (HATEOAS)

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



56

57

## RPC-style Web APIs

- RPC: Remote procedure call**
  - A method or subroutine is executed in another address space, without the programmer explicitly encoding the details of the remote interaction.
- An RPC-style Web API accepts an envelope full of data from its client, and sends a similar envelope back.
  - The method and the scoping information are kept inside the envelope, or on stickers applied to the envelope.
- Every RPC-style Web API defines a brand new vocabulary: method name, method parameters
- Some examples:
  - XML-RPC
  - SOAP.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



57

58

## RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



58

59

## REST (Representational State Transfer)

- **Architectural style proposed by Roy Thomas Fielding.**  
[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
  - Does not define an architecture but requirements for the architecture
- **Representation**
  - Resource-oriented: operates with resources.
    - **Resource:** Any piece of information that can be named. Identified generally by URL
- **State:**
  - value of all properties of a resource at the certain moment.
- **Transfer:** State can be transferred
  - Clients can:
    - 1) retrieve the state of a resource and
    - 2) modify the state of the resource
  - UNIFORM interface

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



59

# REST APIs

- CRUD
  - Most extended approach. Majority of Web APIs nowadays
  - Not follow strictly REST principles
    - More on this next lecture
- Hypermedia
  - Follows strictly REST principles



# Twitter API

Developer

Use cases

Products

Docs

More

GET statuses/home\_timeline

Get Tweet timelines

Curate a collection of Tweets

Optimize Tweets with Cards

Search Tweets

Filter real-time Tweets

Sample real-time Tweets

Get batch historical Tweets

Rules and filtering

Data enrichments

Tweet objects

Tweet compliance

Tweet updates

Direct Messages

Media

Trends

Geo

Ads

Metrics

Publisher tools & SDKs

Twitter for Websites

Developer utilities

API reference index

GET statuses/home\_timeline

Return a collection of the most recent Tweets and Retweets posted by the authenticating user and the users they follow. The home timeline is central to how most users interact with the Twitter service.

Up to 500 Tweets are obtainable on the home timeline. It is more volatile for users that follow many users or follow users who Tweet frequently.

See Working with Timelines for instructions on traversing timelines efficiently.

Resource URL

[https://api.twitter.com/1.1/statuses/home\\_timeline.json](https://api.twitter.com/1.1/statuses/home_timeline.json)

Resource Information

Response formats

JSON

Requires authentication?

Yes (user context only)

Rate limited?

Yes

Requests / 15-min window (user auth)

15

Parameters

Name	Required	Description	Default Value	Example
count	optional	Specifies the number of records to retrieve. Must be less than or equal to 500. Defaults to 20. The value of count is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the count has been applied.	5	
since_id	optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.		12345
max_id	optional	Returns results with an ID less than (that is, older than) or equal to the specified ID.		54321

<https://developer.twitter.com/en/docs.html>



62

## What about current Web applications (RPC or CRUD)?

- Need excessive documentation
  - Exhaustive description of required protocol: HTTP methods, URLs ...
- Integrating a new API inevitably requires writing custom software
  - Similar applications required totally different clients
- When an application API changes, clients break and have to be fixed
  - For instance a change in the object model in the server or the URL structure => change in the client.
- Clients need to store a lot of information
  - Protocol semantics
  - Application semantics

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



62

63

## Web vs Programmable Web

- The **Programmable Web** use the same technologies and communication protocols as the WWW in order to cope with current problems.
- Current differences
  - The data is not delivered necessarily for human consumption (M2M)
  - Nowadays an **specific client** is needed per application at least until we solve the problems derivated from the **semantic challenge**
  - A client can be implemented using any programming language
    - Data is encapsulated and transmitted using any serialization languages such as **JSON, XML, HTML, YAML**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



63

## Hypermedia driven Web APIs

- Follows strictly Fielding dissertation principles.
  - REST APIs must be hypertext driven:  
<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>
- Uses Hypermedia as the Engine of the Application State
  - Hypermedia describes the actions that you can perform with the resources.
    - Client does not memorize operations nor workflow. Everything is in the messages
- Documentation reduced drastically: messages are documented by themselves
  - A REST API should spend almost all of its descriptive effort in defining the media type used for representing resources and driving application states
- Easier to create general clients
  - Example: RSS and Atom PUB. Multiple clients can read the same RSS feed.