

# 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

# The World Wide Web

# What is the World Wide Web?



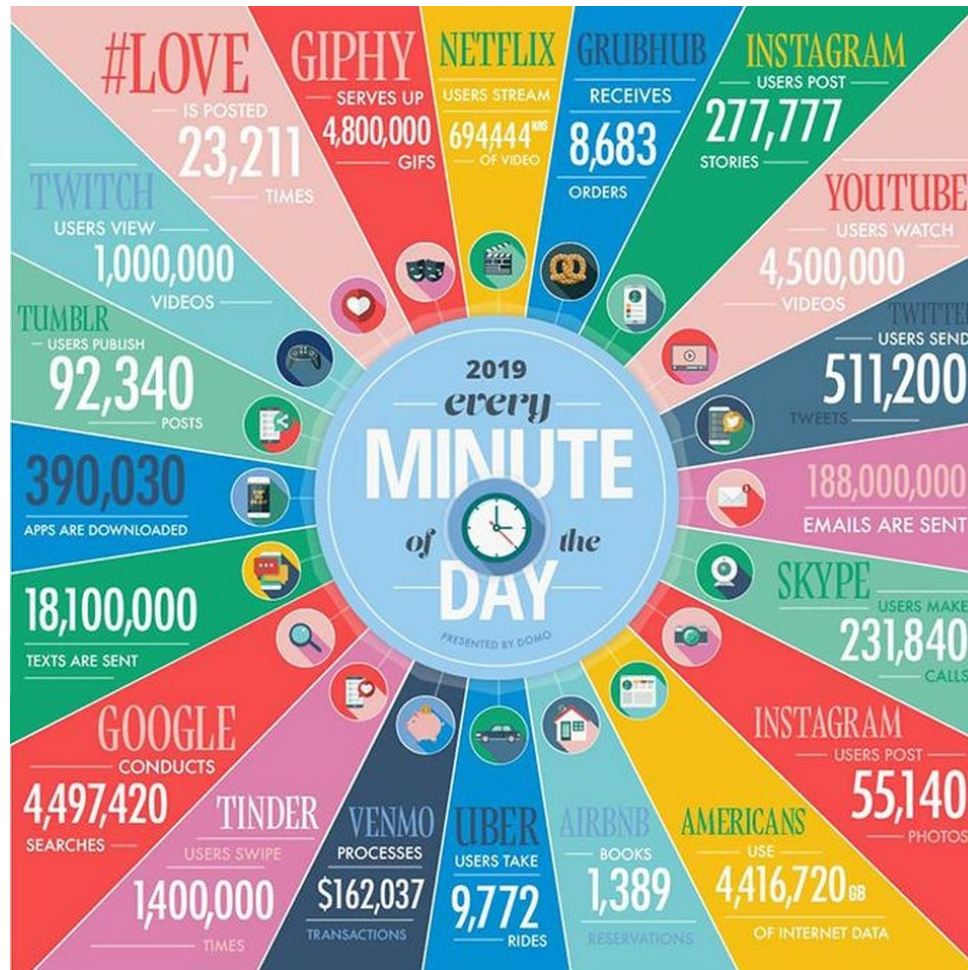
[https://www.youtube.com/watch?v=OM6XIIcm\\_go&start=18&end=190&autoplay=1](https://www.youtube.com/watch?v=OM6XIIcm_go&start=18&end=190&autoplay=1)

# What is the World Wide Web?

## Goal: Distribute data

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

# World Wide Web success. Scalability



Source (2019): <https://www.domo.com/learn/data-never-sleeps-7>

Web is distributed

Web is massively  
decoupled

Web is dynamic

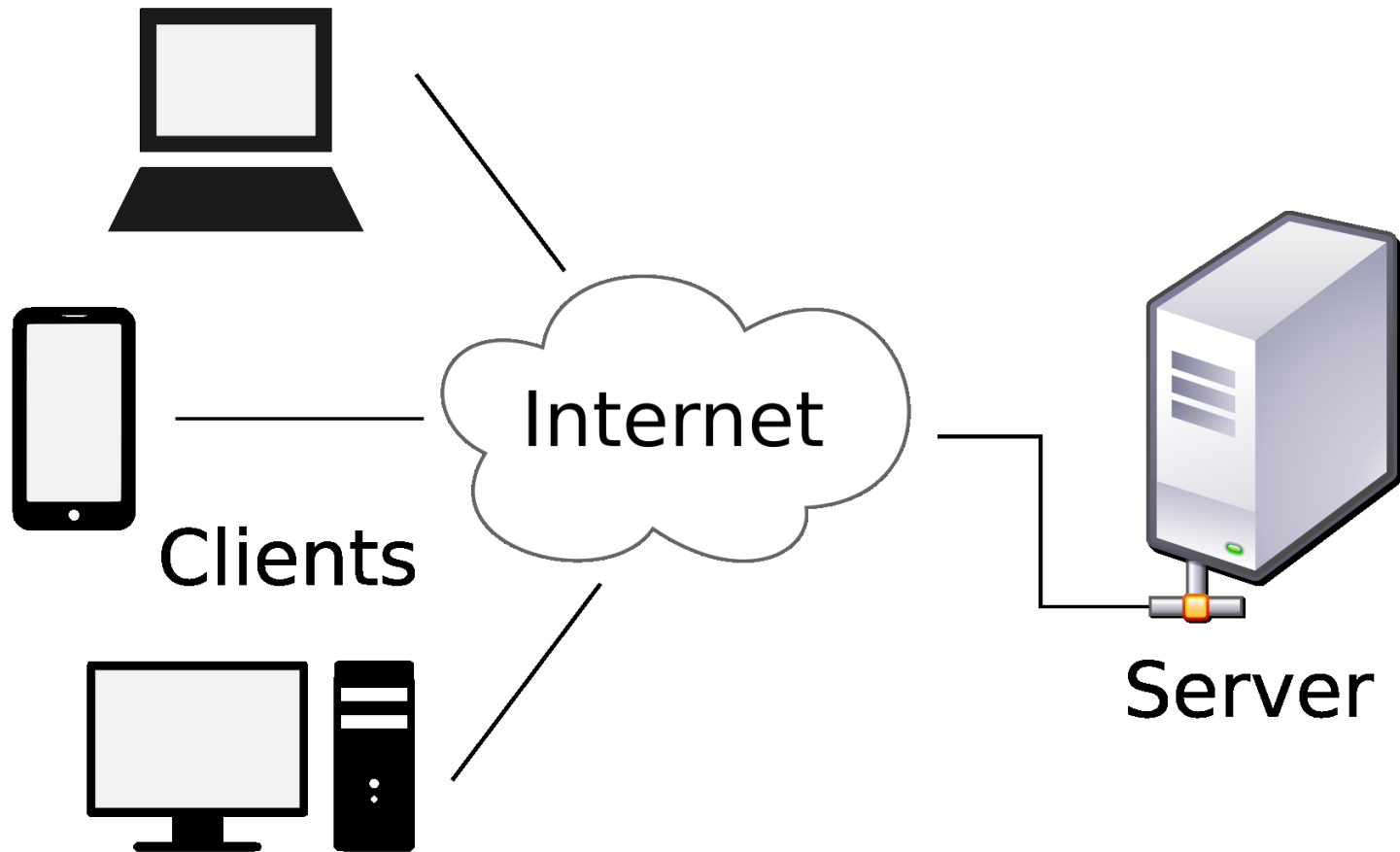
# How the WWW works?

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

# TECHNOLOGIES FOR THE WWW

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

# Client server model



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

# BACKEND

# 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

# DATABASES

# 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

# ACID properties

- Atomicity

- Each transaction is atomic.
- If one part of the transaction fails the whole transaction fails and the database is not modified.

- Consistency

- Databases moves from one valid state to another valid state in each transaction.
- A state is valid if meets all the constraints

- Isolation

- Concurrent access is processed as serial access.
- Not completed transactions might not be visible to other users

- Durability

- Once a transaction is committed it remains in the db.

# 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

# RDBMS Concepts

- **CRUD**

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

- **ORM (Object relational mapping)**

- To access a relational database from an object oriented language 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)**.

# 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

# 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}});
```

# Examples – Non-relational (I)

- As the usage of many popular sites (Amazon, Facebook, Google) increased, research on more adequate models to store and retrieve distributed data became necessary.
- Non-relational databases usually offer better scalability and performance by not supporting all the functionality of a generic relational DBMS
- Examples
  - Dynamo (Amazon): distributed key-value model
  - BigTable (Google): designed to scale across multiple servers
    - <http://research.google.com/archive/bigtable.html>
    - Index (triplet)-value model
  - Cassandra (Facebook): Dynamo implementation
    - <http://cassandra.apache.org/>
  - Hbase
  - Hypertable (Baidu): Inspired in BigTable
  - Hadoop: Distributed storage and processing

# SQL (I)

- SQL (Structured Query Language) is a programming language for managing data in relational DBMS.
- It was created to access the first version of the relational database defined by Codd (see slide 13).
- SQL is an ANSI standard from 1986, and an ISO standard from 1987
  - [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=45498](http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498)
- SQL standard has grown a lot since then.
  - Not all DBMSs implement the whole standard but
  - They all do implement basic functionality for **Data Manipulation** (CRUD) given by the standard commands:
    - INSERT (Create)
    - SELECT (Read)
    - UPDATE (Update)
    - DELETE (Destroy)

## SQL (II)

- SQL allows **Data Definition** with statements like
  - CREATE Table
  - ALTER Table
  - DROP Table
- SQL allows **Data Control** to manage user access with the commands:
  - GRANT (add permissions)
  - REVOKE (remove permissions)

# SQL (III)

- SQL Query

- Executed using command SELECT
- Retrives data, based on any criteria given by CLAUSES (FROM, WHERE, ORDER BY)

```
SELECT FirstName FROM names_table  
WHERE Surname = 'Smith'
```

- Simple SQL statements syntax:

- Usually starts with the desired action – COMMAND
- Then, a CLAUSE with the target
- Finally, a series of CLAUSES may give additional instructions

```
DELETE * FROM names_table  
WHERE Surname = 'Smith'
```

# TRANSPORT PROTOCOL: HTTP

# HTTP

- **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

# HTTP Request parts

- **HTTP method.**
  - Indicates how the client expects the server to process the request.
- **Path**
  - Portion of the URI to the right of the hostname.
  - In terms of the envelope metaphor, the path is the address on the envelope.
- **Request headers**
  - Are key-value pairs of metadata. They are like stickers for the envelope.
  - Can include general headers, request specific headers and entity headers
  - There's a standard list of HTTP headers and applications can define their own.
- **Entity-body**
  - The resource representation.
  - For GET, HEAD and DELETE methods, the entity body is empty. The information needed to complete the request is in the path and the headers.

# 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.

# HTTP Response parts (I)

- **HTTP response code**

- Informs the client about the status of the request process, and how the client should regard this envelope and its contents.
- 3 digit integer followed by a reason phrase (textual description of the code)
- Clearly defined by RFC2616

- **Response headers**

- Same function as the request headers.
- Includes general headers, response specific headers and entity headers.

- **Entity-body**

- Is the resource representation.
- The entity-body is the fulfillment of the HTTP request.

# 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").

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 *response headers*: general, response and entity headers

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

# 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

# HTTP methods (I)

## Defined in RFC2616

- **GET**

- Retrieves the information (in the form of an entity) identified by the Request-URI.
  - Eg. when we access a web site, we get the associated html page.

- **HEAD**

- Identical to GET except that the server returns only headers information in the response (**no** message-body).
  - Used for testing hypertext links for validity, accessibility, and recent modifications.

- **POST**

- Requests that the enclosed entity is stored as a new **subordinate** of the resource identified by the Request-URI.
- Example functions:
  - Adding annotations to existing resources
  - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
  - Providing a block of data, such as the result of a form submit, to a data-handling process
  - Extending a database through an append operation.

# HTTP methods (II)

- **PUT**

- Requests that the enclosed entity is stored under the supplied Request-URI
- Two options:
  - If the Request-URI refers to an already existing entity, the enclosed one should be considered as a modified version of the one residing on the server.
  - Otherwise, the origin server creates one entity with that URI.

- **DELETE**

- Requests that the server delete the resource identified by the Request-URI.

- **OPTIONS**

- Represents a request for information about the communication options available on the request / response chain identified by the Request-URI.
  - Check which are the methods that a resource supports.

# HTTP and REST style

- **HTTP** protocol conforms with REST style and provides a stable “transport” protocol for Web service message exchange.
  - HTTP provides client-server communication style
  - Provides uniform interface with the HTTP methods
  - Is stateless by definition
- It is possible to apply REST concepts to other protocols and systems
  - E.g. Stateless interaction with an FTP site

# DATA SERIALIZATION LANGUAGES

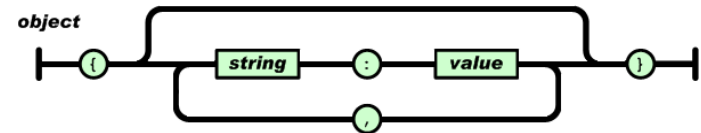
# JSON and XML

- Formats used for representing data that are heavily used to share data among heterogeneous peers
  - Text format (not binary)
  - Language independant
- 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.

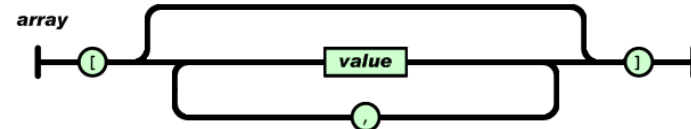
# JSON

- 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  
    }  
}}
```

# 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>)

# 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>

# 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>
```

# Hypermedia (Collection+JSON)

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

Link: <http://amundsen.com/media-types/collection/>

```
{ "collection":
  {
    "version" : "1.0",
    "href" : "http://www.youtypeitwepostit.com/api/",
    "items" : [
      { "href" : "http://www.youtypeitwepostit.com/api/messages/21818525390699506",
        "data" : [
          { "name" : "text", "value" : "Test." },
          { "name" : "date_posted", "value" : "2013-04-22T05:33:58.930Z" }
        ],
        "links" : []
      },
      { "href" : "http://www.youtypeitwepostit.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 FORMATSIN APPENDIX 3: Hypermedia formats

# Hypermedia (XML and JSON)

- XML or JSON as such are not hypermedia (no define hypermedia controls!!!!)
- **XML** contains some extensions to convert an XML document in hypermedia:

- xlink from <http://www.w3.org/1999/xlink>
  - Define a set of attributes to establish relations

```
<book title="Harry Potter">
  <description xlink:type="simple" xlink:href="/images/HPotter.gif" xlink:show="new">
    As his fifth year at Hogwarts School of Witchcraft and Wizardry approaches, Harry is ...
  </description>
</book>
```

- atom:link from <http://www.w3.org/2005/Atom>
  - XML element to establish relations

- **JSON-LD (JSON for Linked Data)** is a media type compatible with JSON that integrates hypermedia controls
  - <http://json-ld.org/>

```
{ "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John\_Lennon",
  "name": "John Lennon",
  "spouse": "http://dbpedia.org/resource/Cynthia\_Lennon" }
```

# Pure hypermedia formats

- HAL
- Siren
- Collection + JSON
- Hydra

MORE IN APPENDIX 3: Hypermedia formats

# CLIENTS

# Types of clients

- Human driven clients
  - Decisions made by humans. IMPORTANT: 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.

# 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
  - **WebSocket**-> Computer communication protocol over TCP that provides full-duplex communication. It enables for instance, pub/sub.

# How a web browser works?



# How a browser works (I) ?

1. The user inserts a URL on the browser address bar
  - The browser makes an HTTP GET request to such URL
2. The browser parses the HTTP response
  - The body of the response contains an HTML document
  - The HTML document might have embedded CSS stylesheets and javascript code
3. The browser transform the HTML document in a tree structure.
  - The tree can be access and modified using DOM interface (<http://www.w3.org/DOM/>): we refer to this tree as DOM tree.
4. The DOM tree is rendered by the browser.
  - Content is provided by the HTML code while the style is given by the CSS code.
  - When the whole DOM is rendered we can say that the webpage has been loaded.
5. Simultaneously the browser retrieves all external files embedded in the HTML document (images and audio files, external javascript and CSS files ...). Those files are stored in a temporal local storage. The DOM has links to those files.

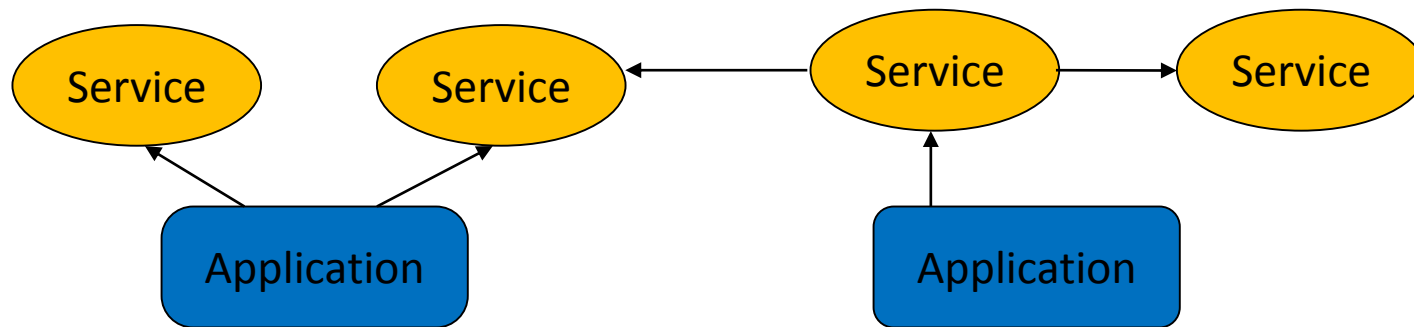
## How a browser works (II) ?

6. The user interacts with the web page shown on the browser.
  - Events are captured and processed by Javascript.
  - Javascript can modify asynchronously the DOM tree, and hence change the content rendered on the browser.
7. Javascript might use AJAX to make asynchronous HTTP requests.
  - HTTP responses are processed and the DOM tree is modified accordingly.
  - The browser keeps the same DOM tree. Just change the corresponding nodes.
8. When the user press a link a new website is loaded in the browser (the process is repeated again).
  - The old DOM tree is deleted.

# SERVICES AND APIS

# Web services

- Web services are logical units that provides certain functionality.
- They are **application independent**
  - services can be used by other services and applications.
  - services can incorporate the functionality of other services (**composite service**)



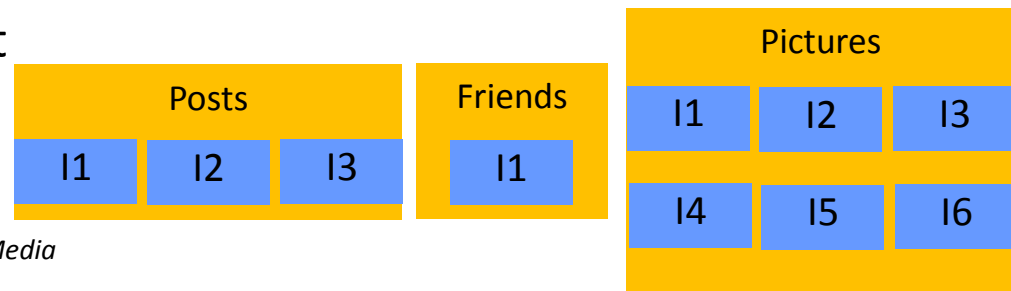
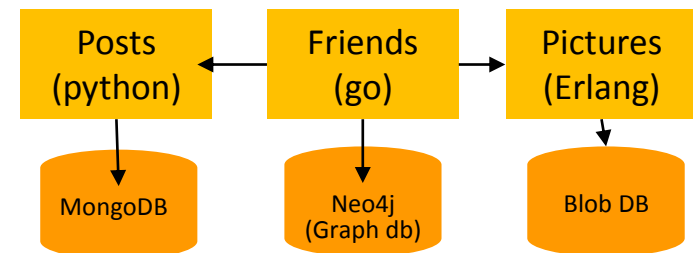
- Services need to communicate to the service consumer:
  - what **functionality** they provide
  - which **data formats** they accept and produce
  - what protocol they use

# Microservices

- Set of small and autonomous services that work together.
  - Business boundaries clear defined -> just a piece of functionality
    - The smaller the better -> microservice should be maintained by small team
  - Each microservice runs in its own OS process.
    - Change independently of each other
    - Must be deploy without a change in the consumer.

- Benefits:

- Technology heterogeneity
- Resilience
- Scaling
- Easy of deployment
- Organizational alignment
- Composability



*Building microservices. Sam Newman. O'Really Media*

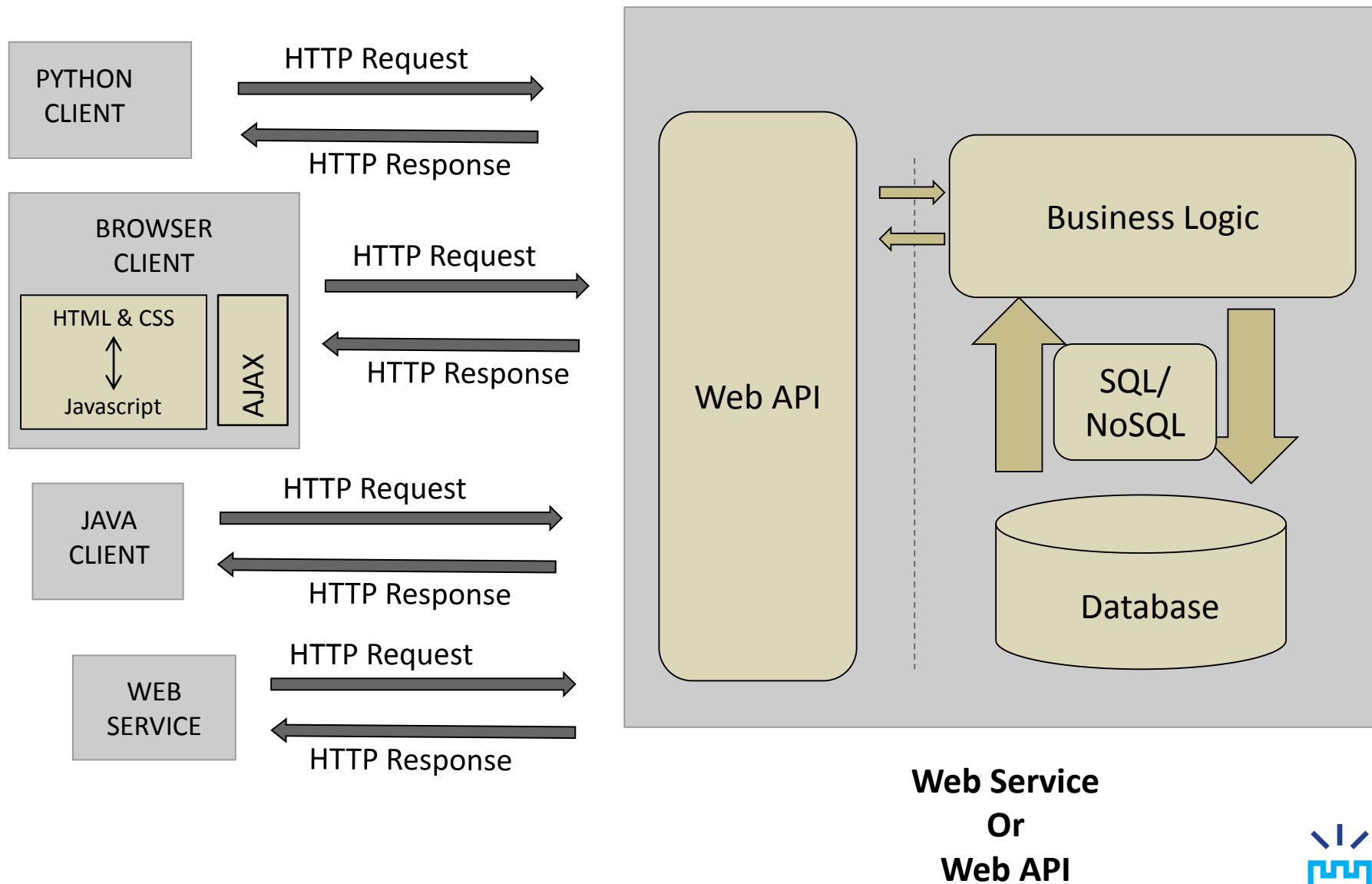
# Web services

- 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.

# APIs and 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**

# Web API



# 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>

## Web APIs Examples.

- **Flickr** Web API can be used to retrieve and upload photos from/to the Flickr sharing service. Pictures can be filtered using multiple criteria.

<https://www.flickr.com/>

<https://www.flickr.com/services/api/>

- **Blurb!** is a web application that makes easy design, publish, market and sell professional-quality books.

<http://www.blurb.com/flickr>

- **Glimmr** is a Flickr viewer for Android. It uses Flickr API to collect data.

<https://play.google.com/store/apps/details?id=com.bourke.glimmr>

- Much more in <http://www.programmableweb.com>

# Architectural styles

- RPC
- REST
  - CRUD
  - Hypermedia (HATEOAS)
- Pub/Sub (Asynchronous Event-Based Collaboration)

# 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.

# RPC

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

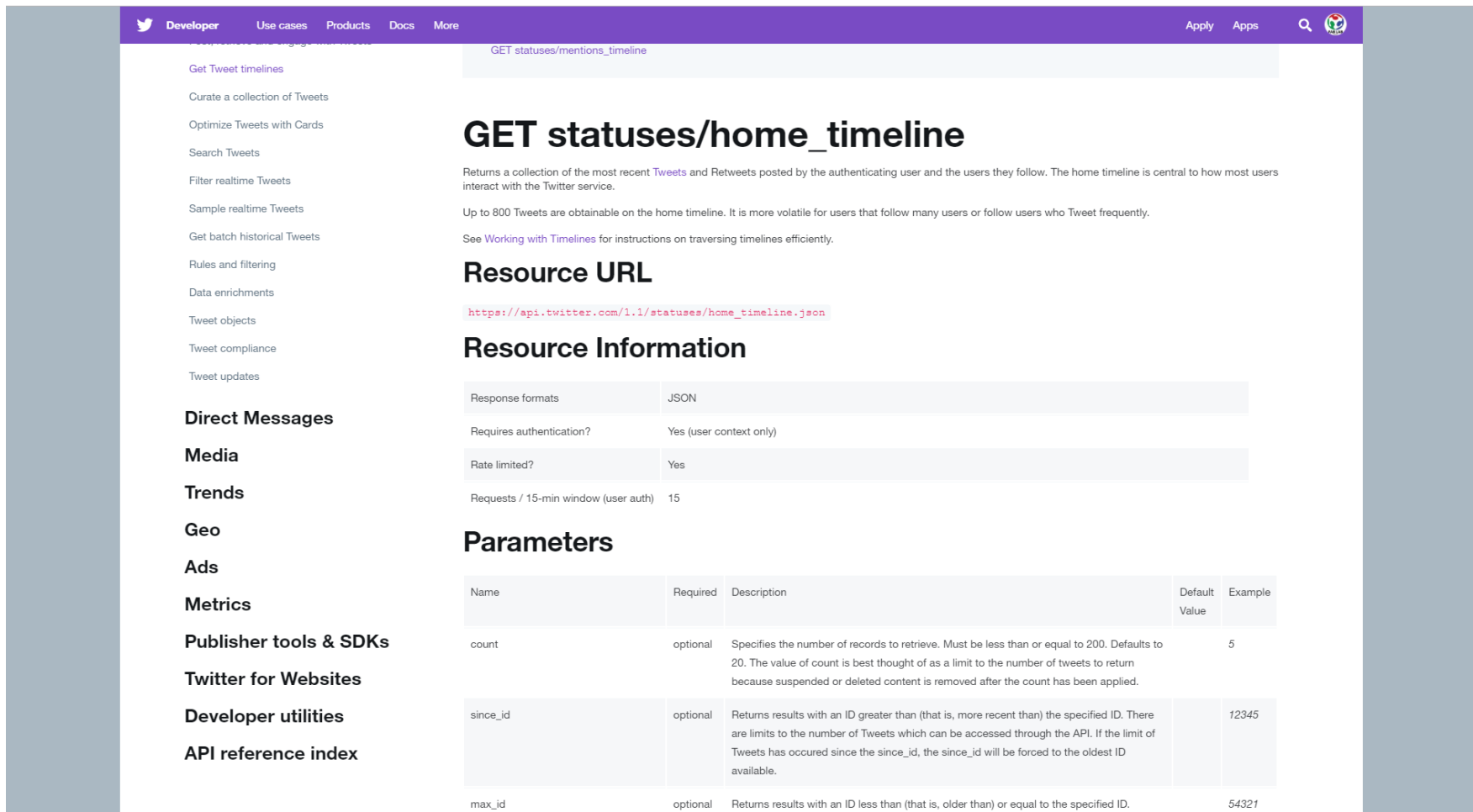
# REST (Representational State Transfer)

- 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

# 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



The screenshot shows the Twitter Developer API documentation for the `GET statuses/home_timeline` endpoint. The page has a purple header with navigation links: Developer, Use cases, Products, Docs, More, Apply, Apps, and a search icon. A left sidebar lists various API categories like Get Tweet timelines, Curate a collection of Tweets, etc. The main content area is titled **GET statuses/home\_timeline** and includes a description of the endpoint, a Resource URL, Resource Information table, and a Parameters table.

**GET statuses/home\_timeline**

Returns 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 800 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`

**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 200. 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 occurred 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>

## Pub / Sub

- Some services emit events (user entered the room)
- Some services are subscribed to those events
  - When the publisher publish the events the subscriber receives the event
- Generally a **broker** is in charge of coordination:
  - Producers publish event to the broker
  - Broker handle subscriptions and inform when an event arrives
- Complex solution BUT creates effective loosely-couple solutions.

# GraphQL

- Mixed of RPC and REST API concepts
  - Created by Facebook.
- GraphQL is a query language APIs, and a server-side runtime for executing queries by using a type system defined for the data.



<https://graphql.org/>  
<https://graphql.org/learn/queries/>

# What about current Web APIs (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

# 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**

# 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.