<WA1/>
<AW1/>
2023

# Web Architecture

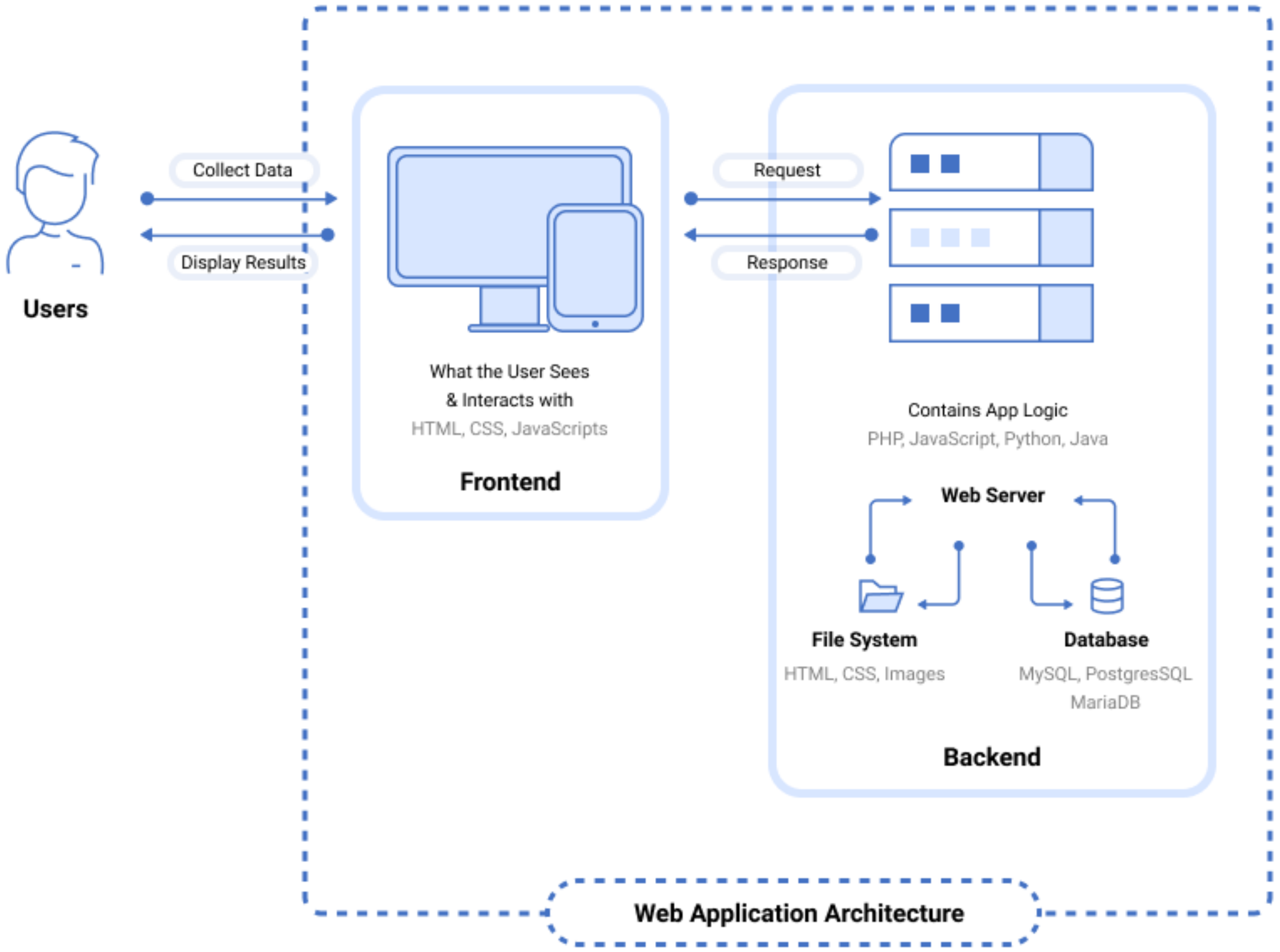
## Layers, Languages, Protocols

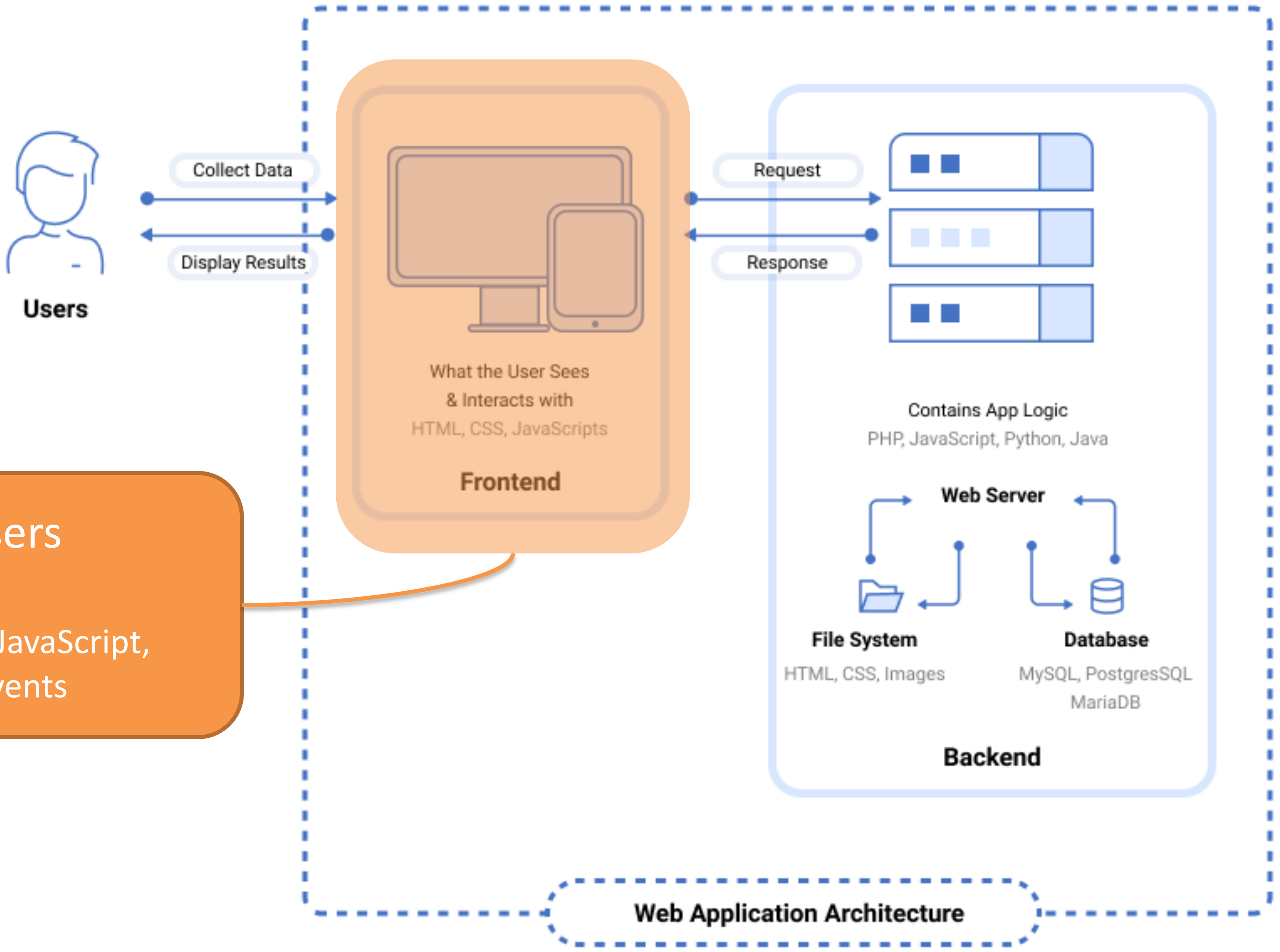
Fulvio Corno
Luigi De Russis
Enrico Masala

## Goal

- Understand what is the Web and its architecture
  - main (logical) components
  - main network protocols
  - existing architectural patterns and languages
- Know the interaction and communication across components
- Learn the basics of how a browser works

- *NOTE: All the topics mentioned here will be presented in more details in the next lectures*

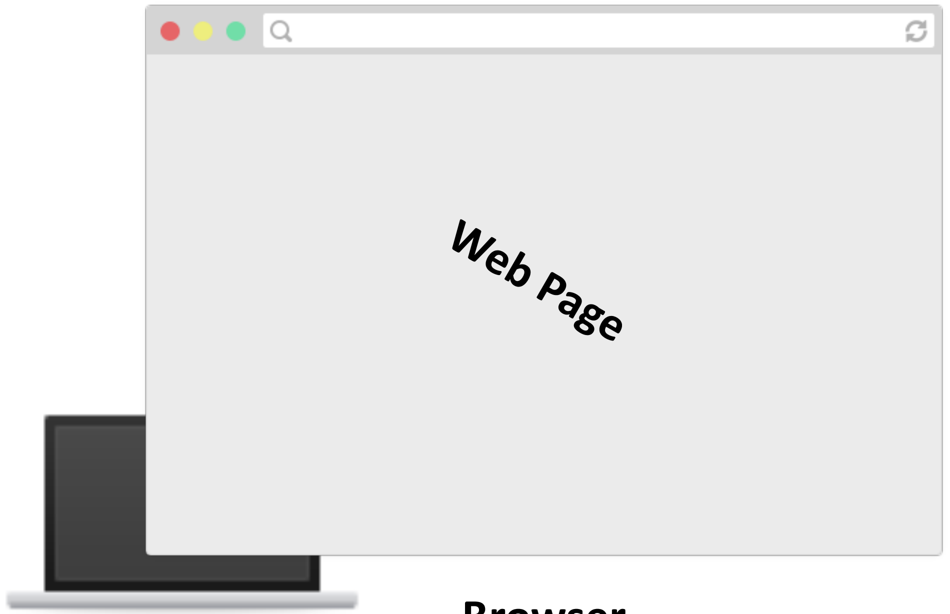

Users
Collect Data
Display Results
What the User Sees
& Interacts with
HTML, CSS, JavaScripts
Frontend
Request
Response
Contains App Logic
PHP, JavaScript, Python, Java
Web Server
File System
HTML, CSS, Images
Database
MySQL, PostgresSQL
MariaDB
Backend
Web Application Architecture



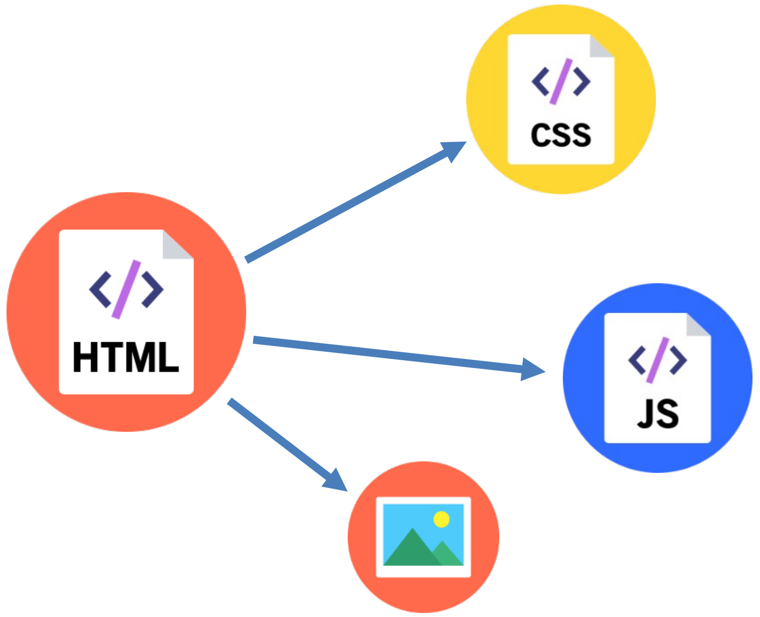
Browsers

HTML 5, CSS, JavaScript, DOM, Events

## Browser



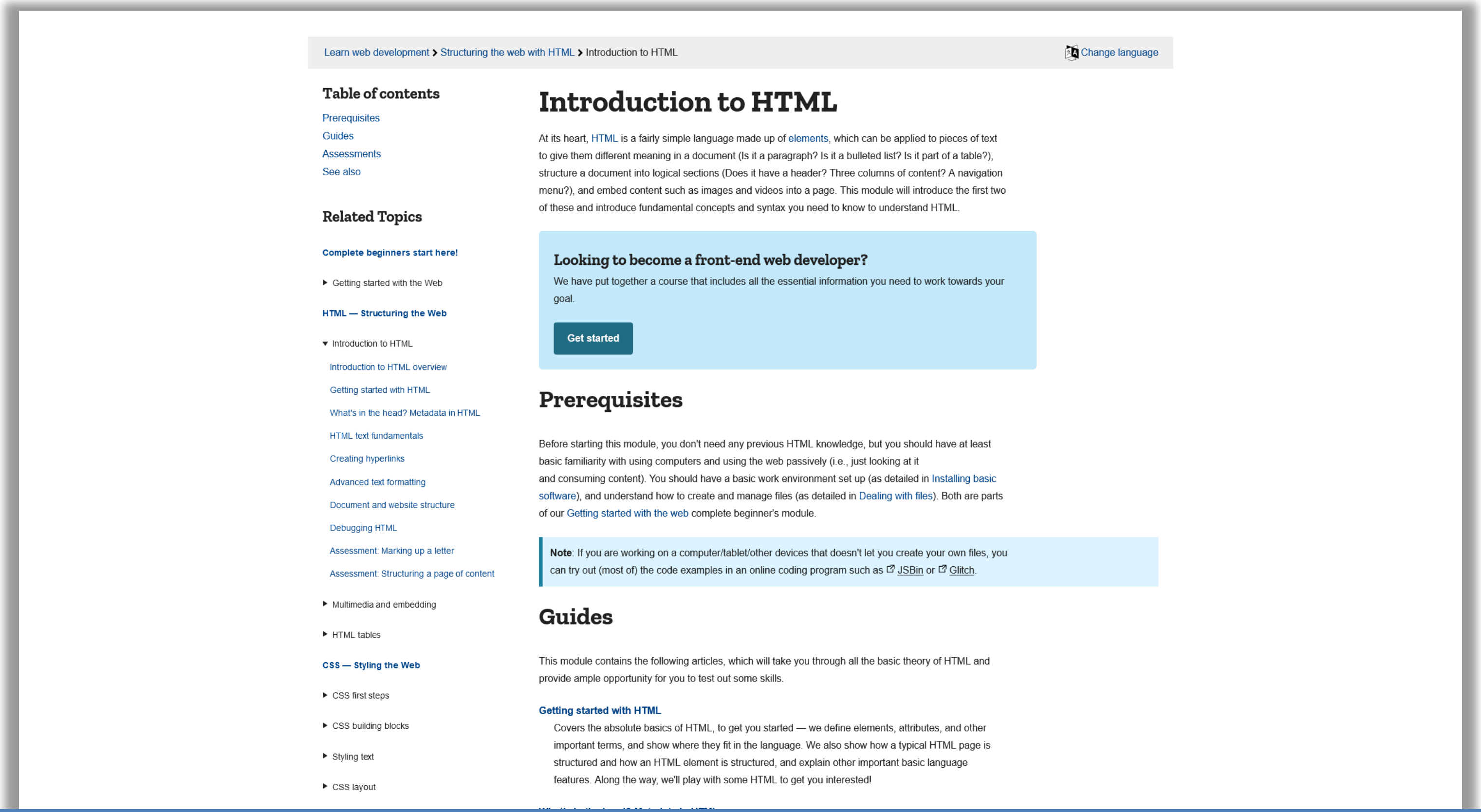
**Browser**



The HTML file might link to other **resources** (images, videos, …)
as well as **JavaScript** and **CSS** files,
which the browser then also loads

These are stored or generated by a **server**

# Quick Introduction to HTML



Learn web development > Structuring the web with HTML > Introduction to HTML

Change language

Table of contents

Prerequisites
Guides
Assessments
See also

Related Topics

Complete beginners start here!

- Getting started with the Web

HTML — Structuring the Web

- Introduction to HTML
  - Introduction to HTML overview
  - Getting started with HTML
  - What's in the head? Metadata in HTML
  - HTML text fundamentals
  - Creating hyperlinks
  - Advanced text formatting
  - Document and website structure
  - Debugging HTML
  - Assessment: Marking up a letter
  - Assessment: Structuring a page of content
- Multimedia and embedding
- HTML tables

CSS — Styling the Web

- CSS first steps
- CSS building blocks
- Styling text
- CSS layout

Introduction to HTML

At its heart, HTML is a fairly simple language made up of elements, which can be applied to pieces of text to give them different meaning in a document (Is it a paragraph? Is it a bulleted list? Is it part of a table?), structure a document into logical sections (Does it have a header? Three columns of content? A navigation menu?), and embed content such as images and videos into a page. This module will introduce the first two of these and introduce fundamental concepts and syntax you need to know to understand HTML.

Looking to become a front-end web developer?

We have put together a course that includes all the essential information you need to work towards your goal.

Get started

Prerequisites

Before starting this module, you don't need any previous HTML knowledge, but you should have at least basic familiarity with using computers and using the web passively (i.e., just looking at it and consuming content). You should have a basic work environment set up (as detailed in Installing basic software), and understand how to create and manage files (as detailed in Dealing with files). Both are parts of our Getting started with the web complete beginner's module.

**Note**: If you are working on a computer/tablet/other devices that doesn't let you create your own files, you can try out (most of) the code examples in an online coding program such as JSBin or Glitch.

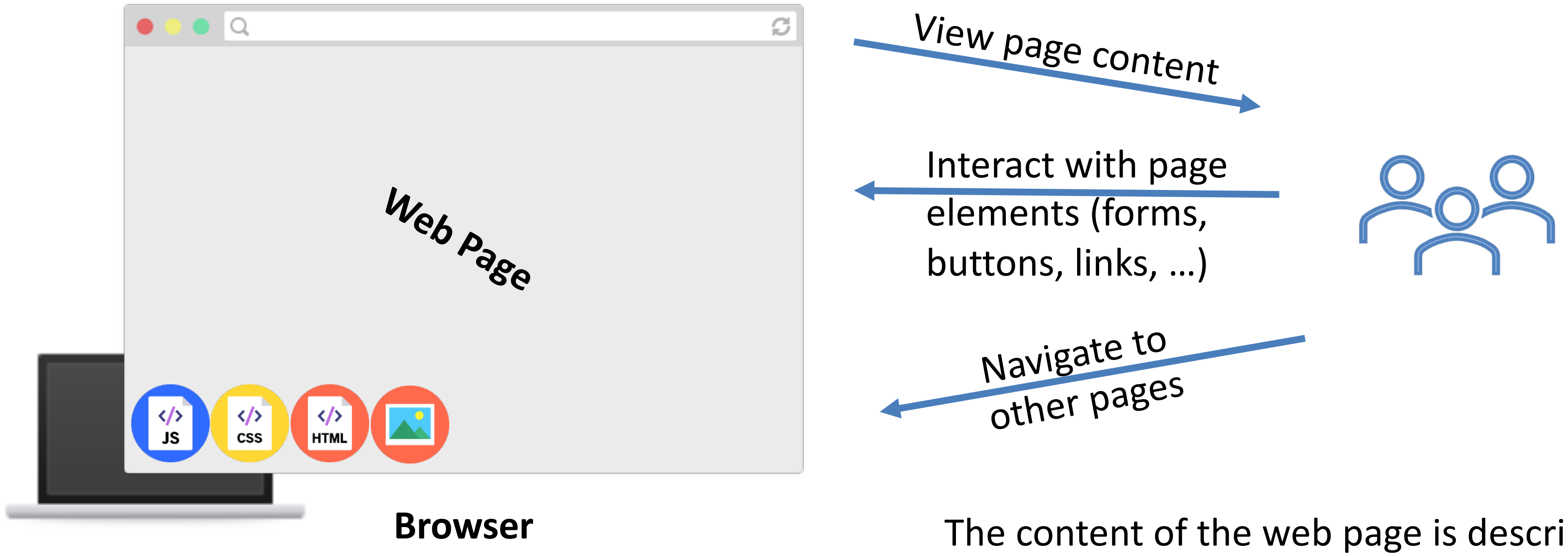
Guides

This module contains the following articles, which will take you through all the basic theory of HTML and provide ample opportunity for you to test out some skills.

Getting started with HTML
Covers the absolute basics of HTML, to get you started — we define elements, attributes, and other important terms, and show where they fit in the language. We also show how a typical HTML page is structured and how an HTML element is structured, and explain other important basic language features. Along the way, we'll play with some HTML to get you interested!

https://developer.mozilla.org/docs/Learn/HTML/Introduction_to_HTML

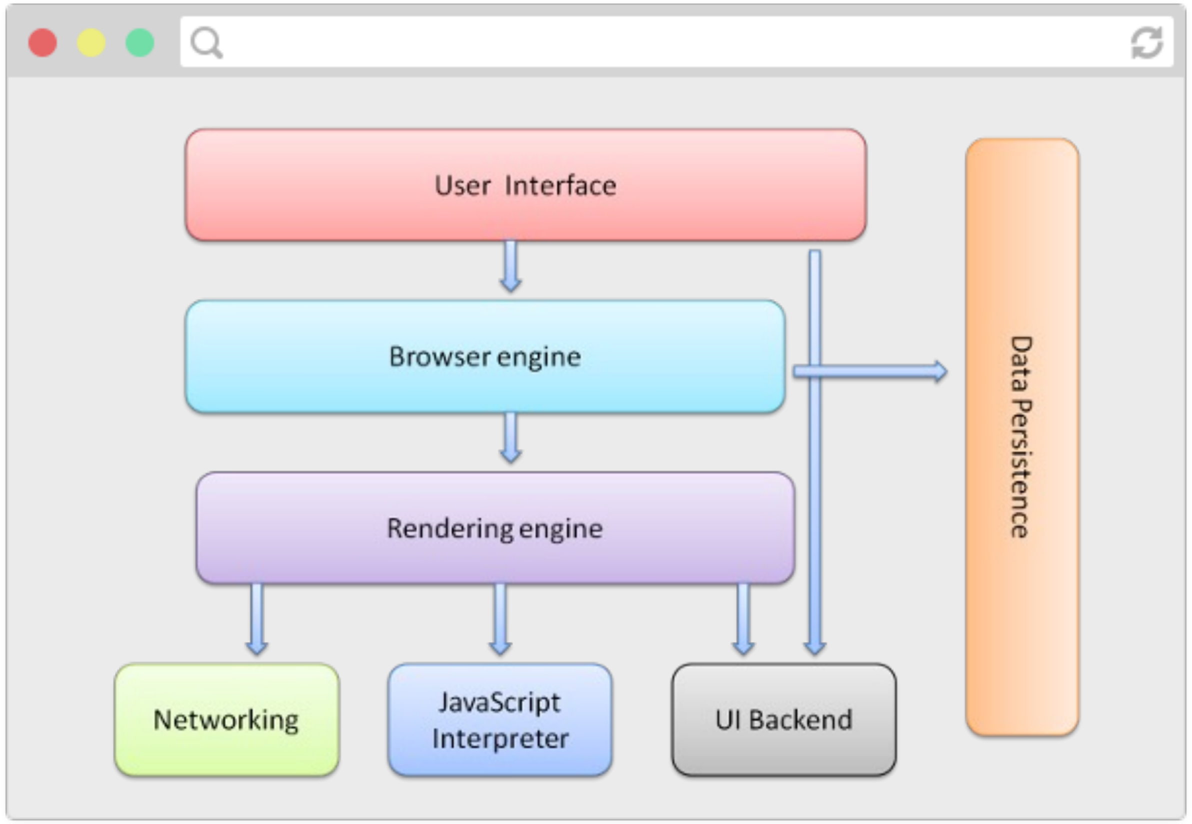
## Browser



The content of the web page is described by HTML+CSS.

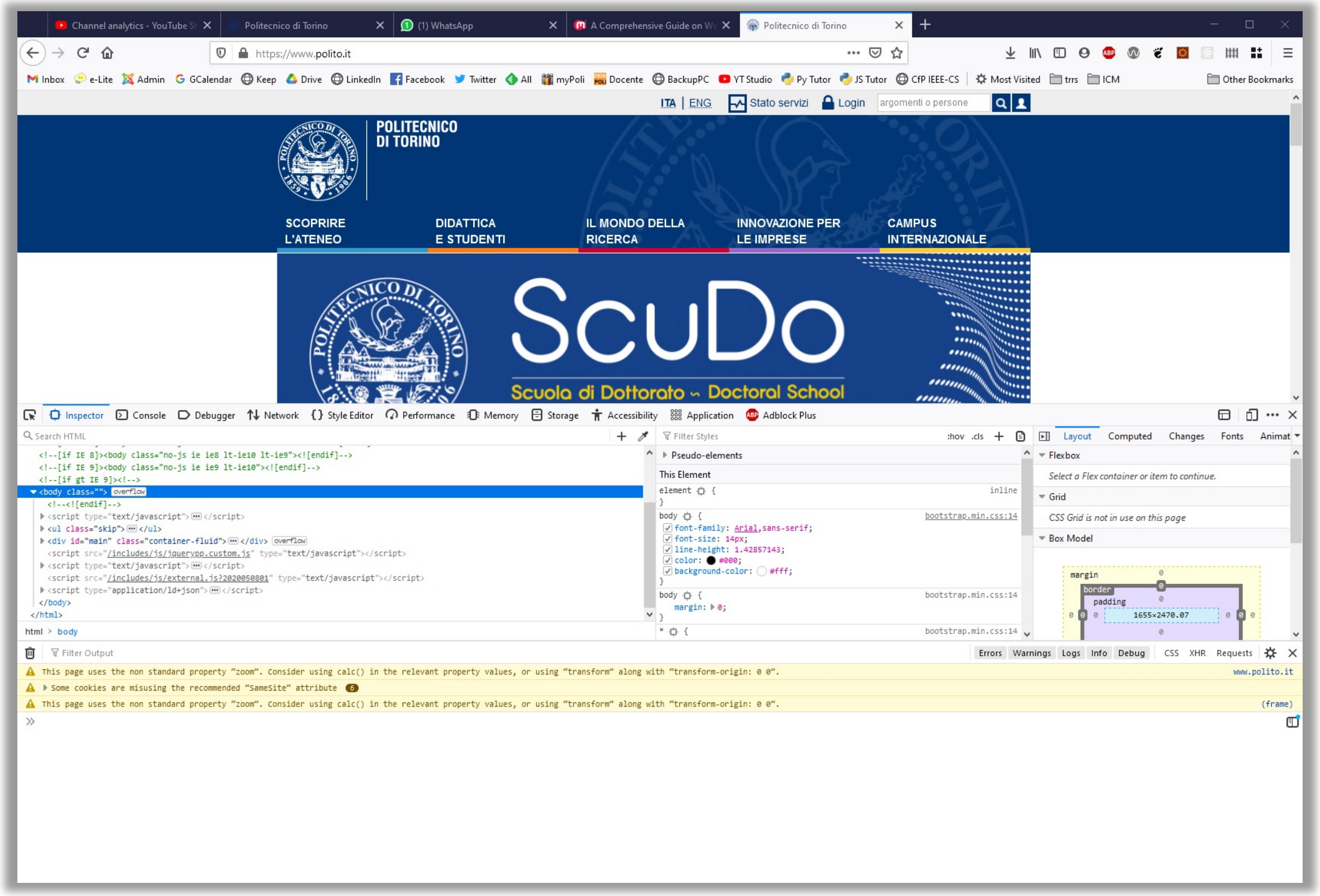
Clicking on a link brings the user to a **new page**.
Interacting with other elements may generate ***Events*** inside the browser.
Such Events are “captured” by JavaScript and may **update the page content**.

# Conceptual Browser Architecture (from 10,000 feet)



- **User Interface**: the address bar, back/forward button, bookmarking menu, etc. Every part of the browser display except the window where you see the requested page
- The **Browser Engine** marshals actions between the UI and the rendering engine
- **Rendering Engine:** responsible for displaying the requested content. For example, if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen
- **Networking**: for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface
- **UI Backend**: used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods
- **JavaScript Interpreter:** used to parse and execute JavaScript code
- **Data Persistence:** a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as LocalStorage, IndexedDB, WebSQL and FileSystem

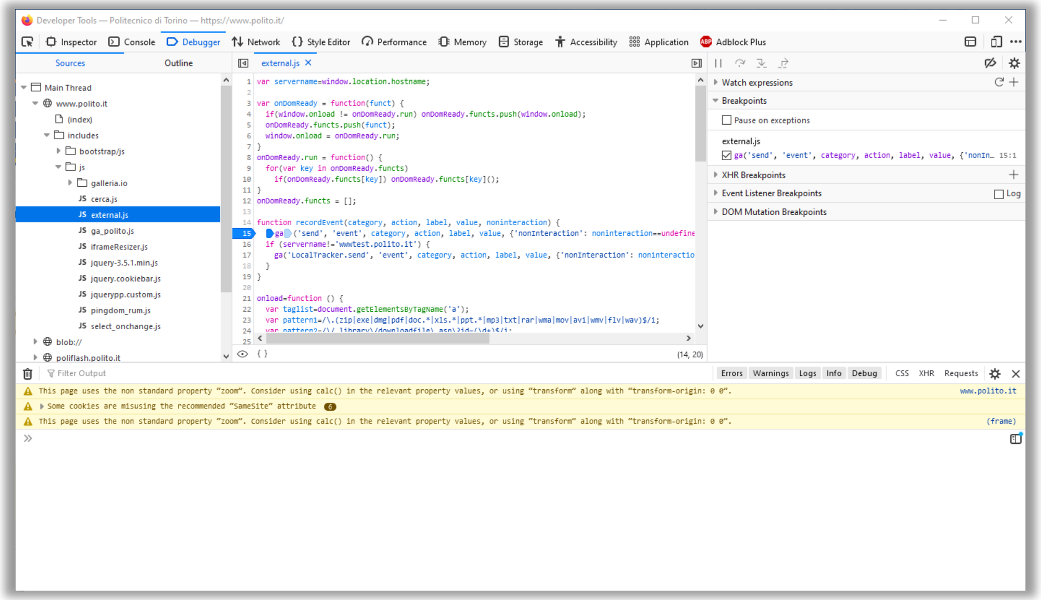
# Browser Development tools



```
<!--[if IE 8]><body class="no-js ie ie8 lt-ie10 lt-ie9"><![endif]-->
<!--[if IE 9]><body class="no-js ie ie9 lt-ie10"><![endif]-->
<!--[if gt IE 9]><!-->
<body class="">
  <!--<![endif]-->
  <script type="text/javascript">…</script>
  <ul class="skip">…</ul>
  <div id="main" class="container-fluid">…</div>
  <script src="/includes/js/jquerypp.custom.js" type="text/javascript"></script>
  <script type="text/javascript">…</script>
  <script src="/includes/js/external.js?2020050801" type="text/javascript"></script>
  <script type="application/ld+json">…</script>
</body>
</html>
```

```
element {
}                                   inline
body {                              bootstrap.min.css:14
  font-family: Arial,sans-serif;
  font-size: 14px;
  line-height: 1.42857143;
  color: #000;
  background-color: #fff;
}
body {                              bootstrap.min.css:14
  margin: 0;
}
```

```
This page uses the non standard property "zoom". Consider using calc() in the relevant property values, or using "transform" along with "transform-origin: 0 0".   www.polito.it
Some cookies are misusing the recommended "SameSite" attribute 6
This page uses the non standard property "zoom". Consider using calc() in the relevant property values, or using "transform" along with "transform-origin: 0 0".   (frame)
```

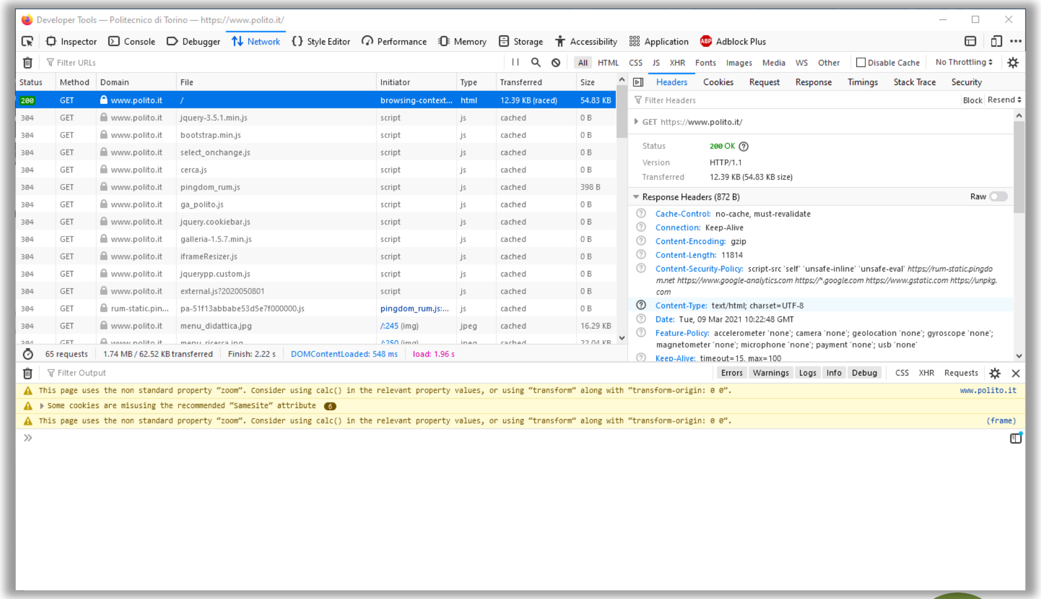


```
var servername=window.location.hostname;

var onDomReady = function(funct) {
  if(window.onload != onDomReady.run) onDomReady.functs.push(window.onload);
  onDomReady.functs.push(funct);
  window.onload = onDomReady.run;
}
onDomReady.run = function() {
  for(var key in onDomReady.functs)
    if(onDomReady.functs[key]) onDomReady.functs[key]();
}
onDomReady.functs = [];

function recordEvent(category, action, label, value, noninteraction) {
  ga('send', 'event', category, action, label, value, {'nonInteraction': noninteraction==undefine
  if (servername!='wwwtest.polito.it') {
    ga('LocalTracker.send', 'event', category, action, label, value, {'nonInteraction': noninteractio
  }
}

onload=function () {
  var taglist=document.getElementsByTagName('a');
  var pattern1=/\.(zip|exe|dmg|pdf|doc.*|xls.*|ppt.*|mp3|txt|rar|wma|mov|avi|wmv|flv|wav)$/i;
```



| Status | Method | Domain | File | Initiator | Type | Transferred | Size |
|---|---|---|---|---|---|---|---|
| 200 | GET | www.polito.it | / | browsing-context… | html | 12.39 KB (raced) | 54.83 KB |
| 304 | GET | www.polito.it | jquery-3.5.1.min.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | bootstrap.min.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | select_onchange.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | cerca.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | pingdom_rum.js | script | js | cached | 398 B |
| 304 | GET | www.polito.it | ga_polito.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | jquery.cookiebar.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | galleria-1.5.7.min.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | iframeResizer.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | jquerypp.custom.js | script | js | cached | 0 B |
| 304 | GET | www.polito.it | external.js?2020050801 | script | js | cached | 0 B |
| 304 | GET | rum-static.pin… | pa-51f13abbabe53d5e7f000000.js | pingdom_rum.js:… | js | cached | 0 B |
| 304 | GET | www.polito.it | menu_didattica.jpg | /:245 (img) | jpeg | cached | 16.29 KB |

65 requests | 1.74 MB / 62.52 KB transferred | Finish: 2.22 s | DOMContentLoaded: 548 ms | load: 1.96 s

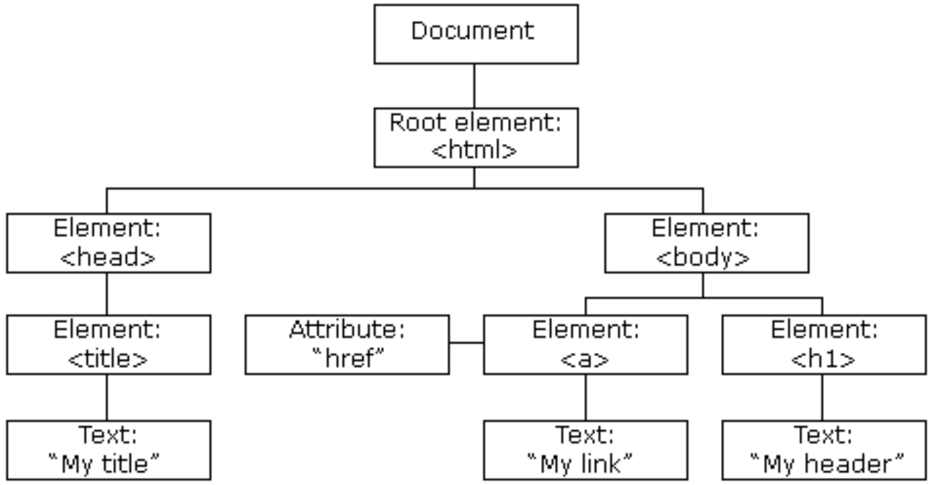
```
GET https://www.polito.it/
Status       200 OK
Version      HTTP/1.1
Transferred  12.39 KB (54.83 KB size)

Response Headers (872 B)
Cache-Control: no-cache, must-revalidate
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 11814
Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval' https://rum-static.pingdom.net https://www.google-analytics.com https://*.google.com https://www.gstatic.com https://unpkg.com
Content-Type: text/html; charset=UTF-8
Date: Tue, 09 Mar 2021 10:22:48 GMT
Feature-Policy: accelerometer 'none'; camera 'none'; geolocation 'none'; gyroscope 'none'; magnetometer 'none'; microphone 'none'; payment 'none'; usb 'none'
Keep-Alive: timeout=15, max=100
```

# Document Object Model (DOM)

"The W3C **Document Object Model** (DOM) is a *platform and language-neutral interface* that allows programs and scripts to dynamically *access* and *update* the content, structure, and style of a document."

- Standard **data structure** for representing the web page content
- Allows to get, change, add, or delete HTML elements
- Supported by all browsers
- **JavaScript programs can read and modify the DOM**
- Abstracts and standardizes APIs to
  - Browser
  - HTML

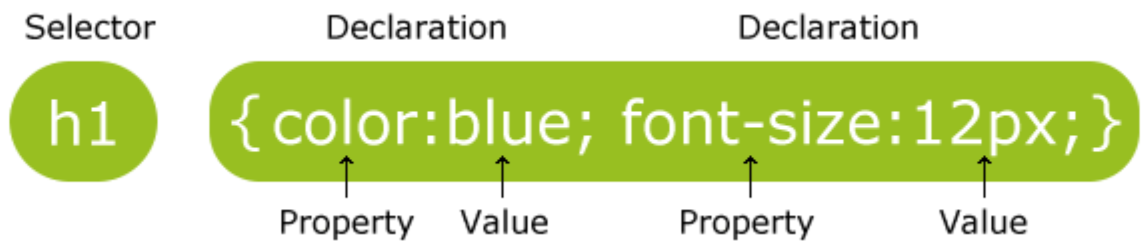


# Cascading Style Sheets (CSS)

- Allow the definition of complex layouts
- Adapt web pages to
  - different resolutions
  - different devices (e.g., smartphones)
  - different preferences (e.g., color schemes)
  - to different media (e.g., text vs. video)
  - in a standard way

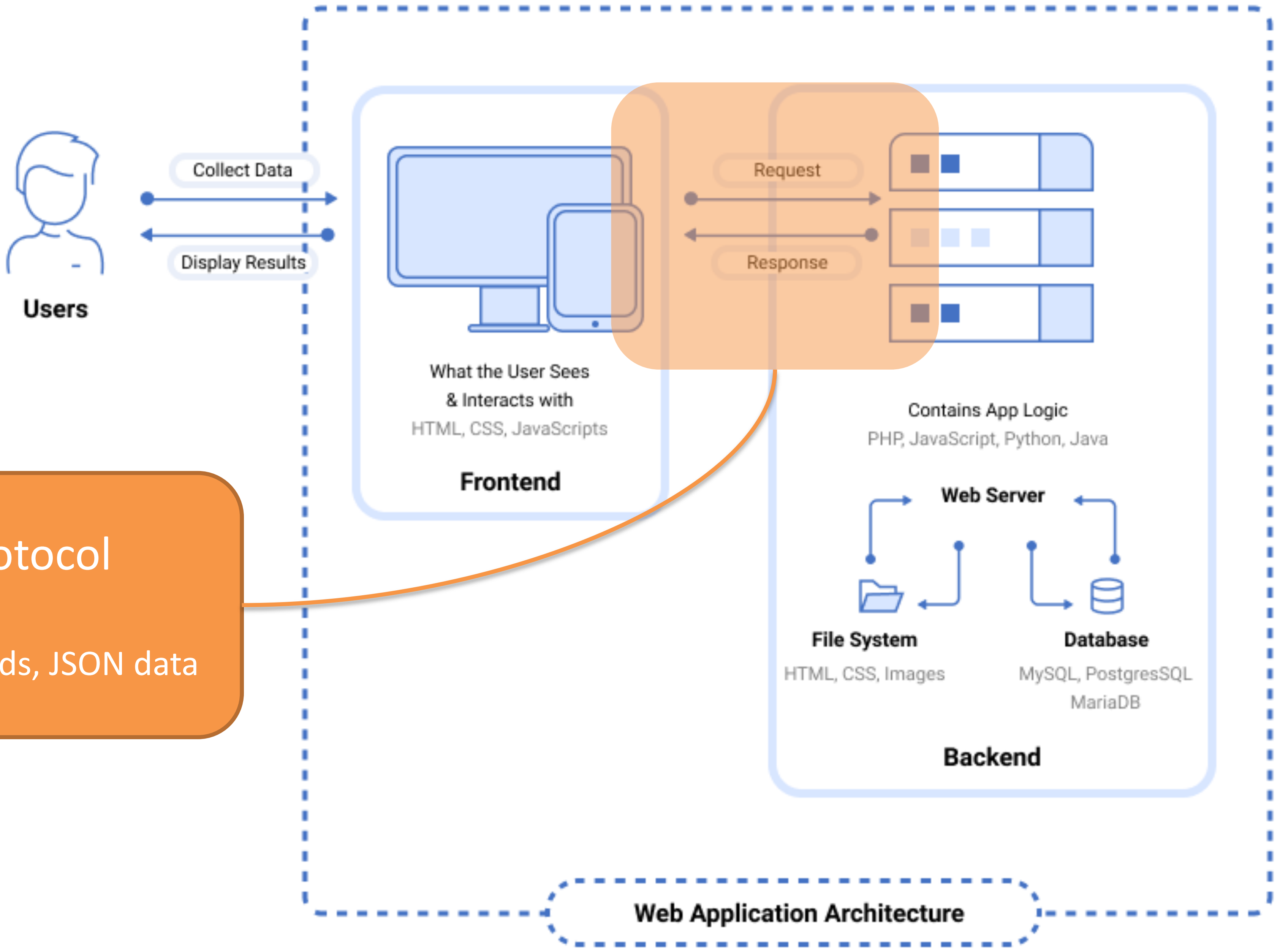
# Cascading Style Sheets (CSS)

- A set of "*declarations*" applied to some "*selectors*"
  - Selectors identify portions of the DOM
  - Declarations set the value of some properties
  - Properties control everything
    - color, size, font, alignment, border, shadow, position, selection status, transitions, links, buttons, cursors, …



# JavaScript

- JS Interpreter Embedded in the Browser
  - Executes within a strict "sandbox"
- JS Scripts loaded by the HTML page
  - `<script src="/js/myscript.js" type="text/javascript"></script>`
- JS Scripts have read-write access to
  - Browser API
  - HTML DOM (including form data)
  - User events and actions



HTTP Protocol

URI, HTTP methods, JSON data

RFC 2616, RFC 2617
http://www.w3.org/Protocols

## HTTP protocol

```
GET / HTTP/1.1
Host: www.polito.it
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive
Cookie: __utma=55042356.701936439.1606736391.1615238467.1615289682.230; __utmz=55042356. [...]
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
```

HTTP Request

# HTTP protocol

RFC 2616, RFC 2617
http://www.w3.org/Protocols

```
GET / HTTP/1.1
Host: www.polito.it
User-Agent: Mozilla/5.0 (Window
Accept: text/html,application/
Accept-Language: en-US,en;q=0
Accept-Encoding: gzip, defla
DNT: 1
Connection: keep-alive
Cookie: __utma=55042356.7
Upgrade-Insecure-Request
Pragma: no-cache
Cache-Control: no-cache
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 09 Mar 2021 14:21:35 GMT
Server: Apache
Strict-Transport-Security: max-age=31536000
Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval' [...]
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Referrer-Policy: no-referrer-when-downgrade
Feature-Policy: accelerometer 'none'; camera 'none'; geolocation 'none'; [...]
Last-Modified: Tue, 09 Mar 2021 14:03:41 GMT
Cache-Control: no-cache, must-revalidate
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 11905
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="it">
<head>
          <meta charset="UTF-8">
          <title>Politecnico di Torino</title>
. . .
```

Header

Blank line

Body

# HTTP Response Body

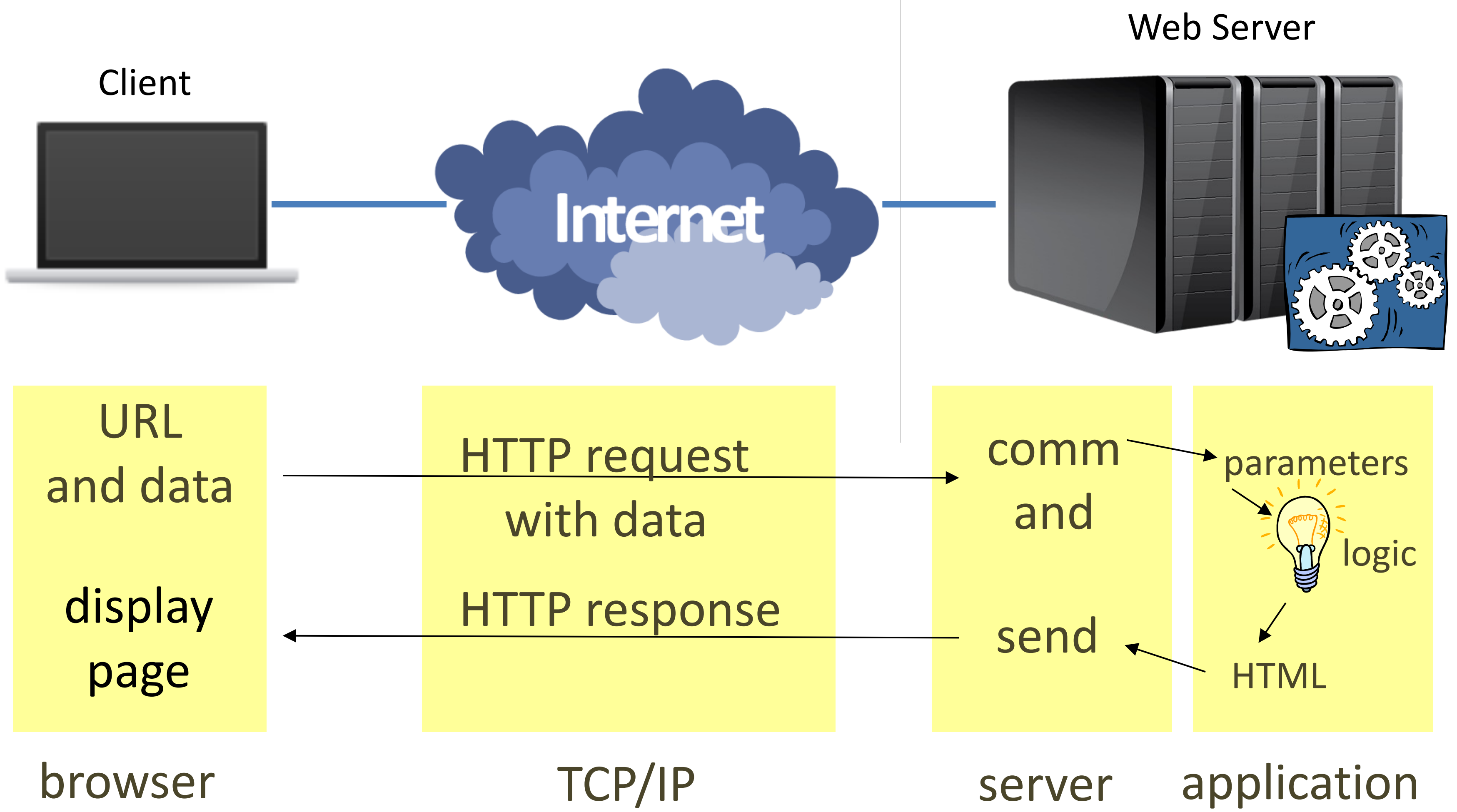
**Generation**

- Empty Response Body
  - Errors
- Static file (exists in the server)
  - HTML (seldom)
  - Images, JavaScript, CSS, …
- Dynamically generated on-the-fly by the server
  - HTML (generated with templates)
  - JSON data

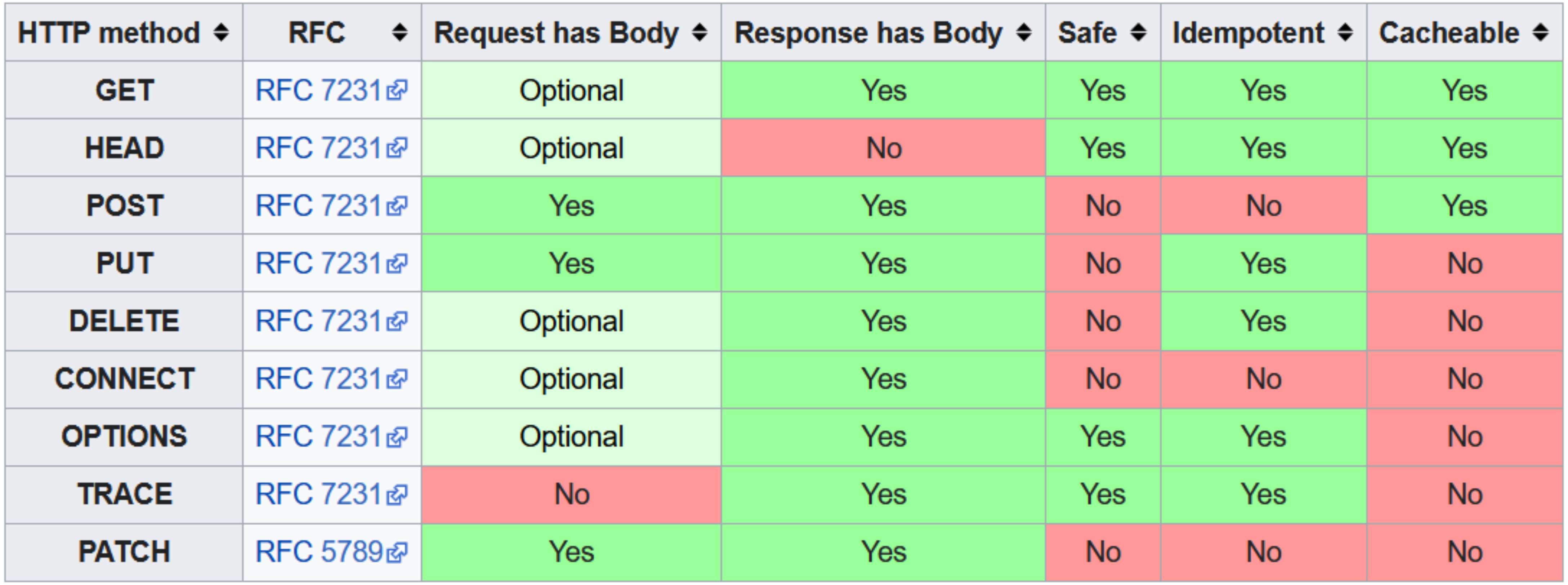
**File and Content Type**

- HTTP does not care about the meaning of the payload
- Web content
  - HTML, CSS, JS
  - Used by the browser
- Data content (API)
  - JSON, XML, binary data, …
  - Used by JavaScript code

# Dynamic Web Transaction

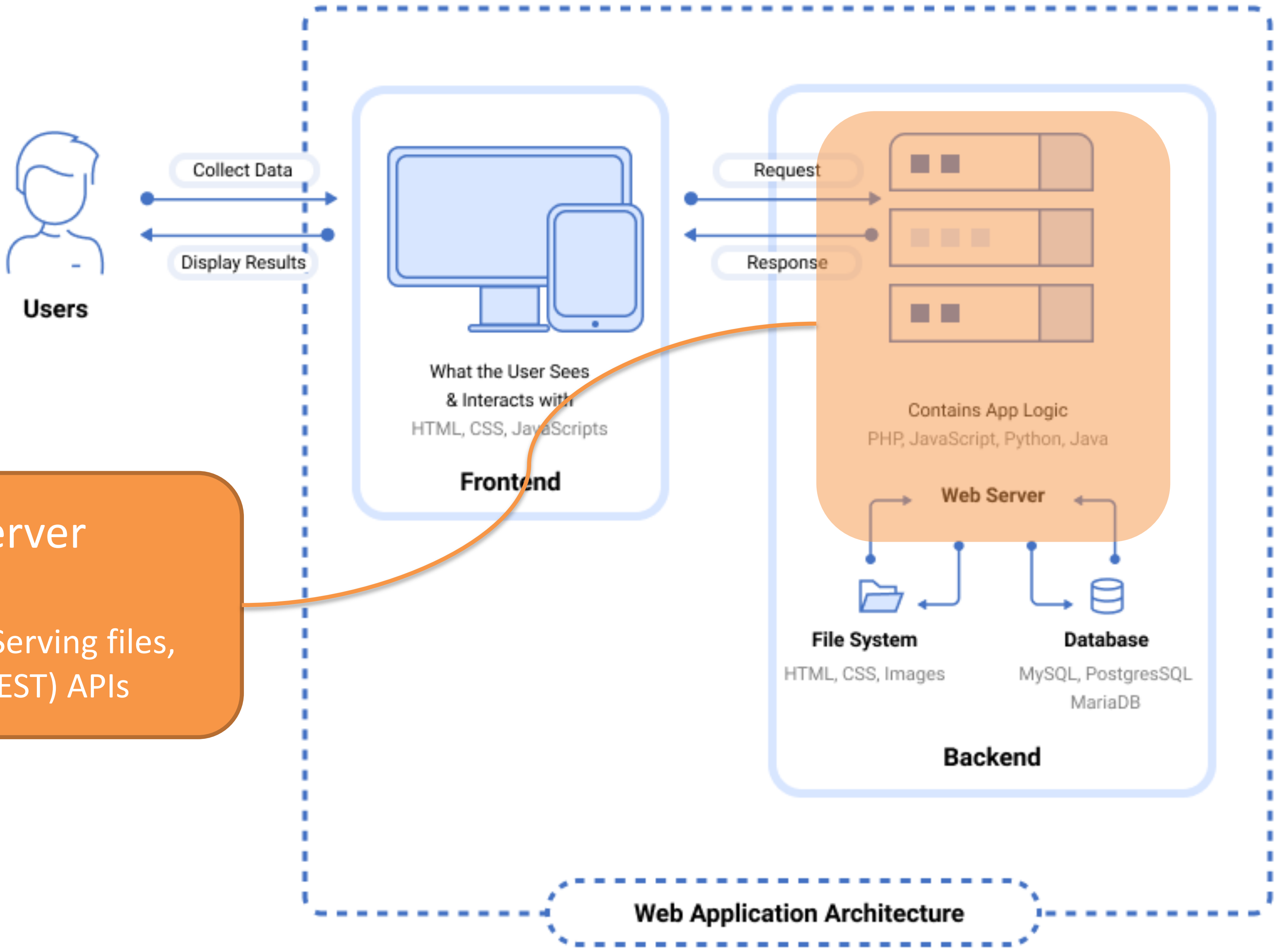


# HTTP Methods



| HTTP method | RFC | Request has Body | Response has Body | Safe | Idempotent | Cacheable |
|---|---|---|---|---|---|---|
| GET | RFC 7231 | Optional | Yes | Yes | Yes | Yes |
| HEAD | RFC 7231 | Optional | No | Yes | Yes | Yes |
| POST | RFC 7231 | Yes | Yes | No | No | Yes |
| PUT | RFC 7231 | Yes | Yes | No | Yes | No |
| DELETE | RFC 7231 | Optional | Yes | No | Yes | No |
| CONNECT | RFC 7231 | Optional | Yes | No | No | No |
| OPTIONS | RFC 7231 | Optional | Yes | Yes | Yes | No |
| TRACE | RFC 7231 | No | Yes | Yes | Yes | No |
| PATCH | RFC 5789 | Yes | Yes | No | No | No |

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods



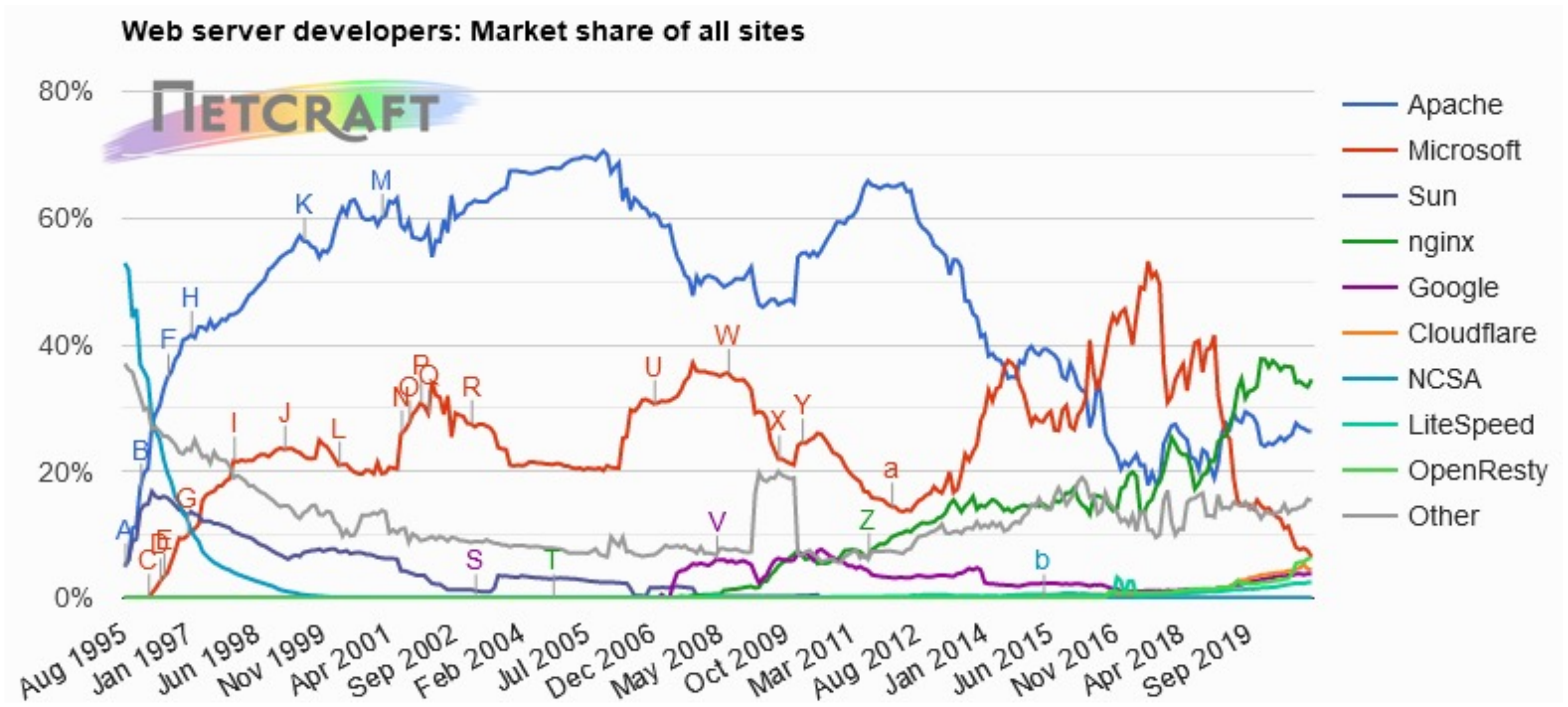
Web Server

Serving pages, Serving files, Executing (REST) APIs

## Web Server

- A web server delivers web resources in response to a request
  - manages the HTTP protocol to handle requests and provide responses
- It either reads or generates a web page
  - receives client requests
  - reads *static page* from the filesystem
  - asks the application server to generate *dynamic pages* (server-side)
  - provides a file (HTML, CSS, JS, JSON, …) back to the client
- One HTTP connection for each request
- Multi-process, multi-threaded or process pool

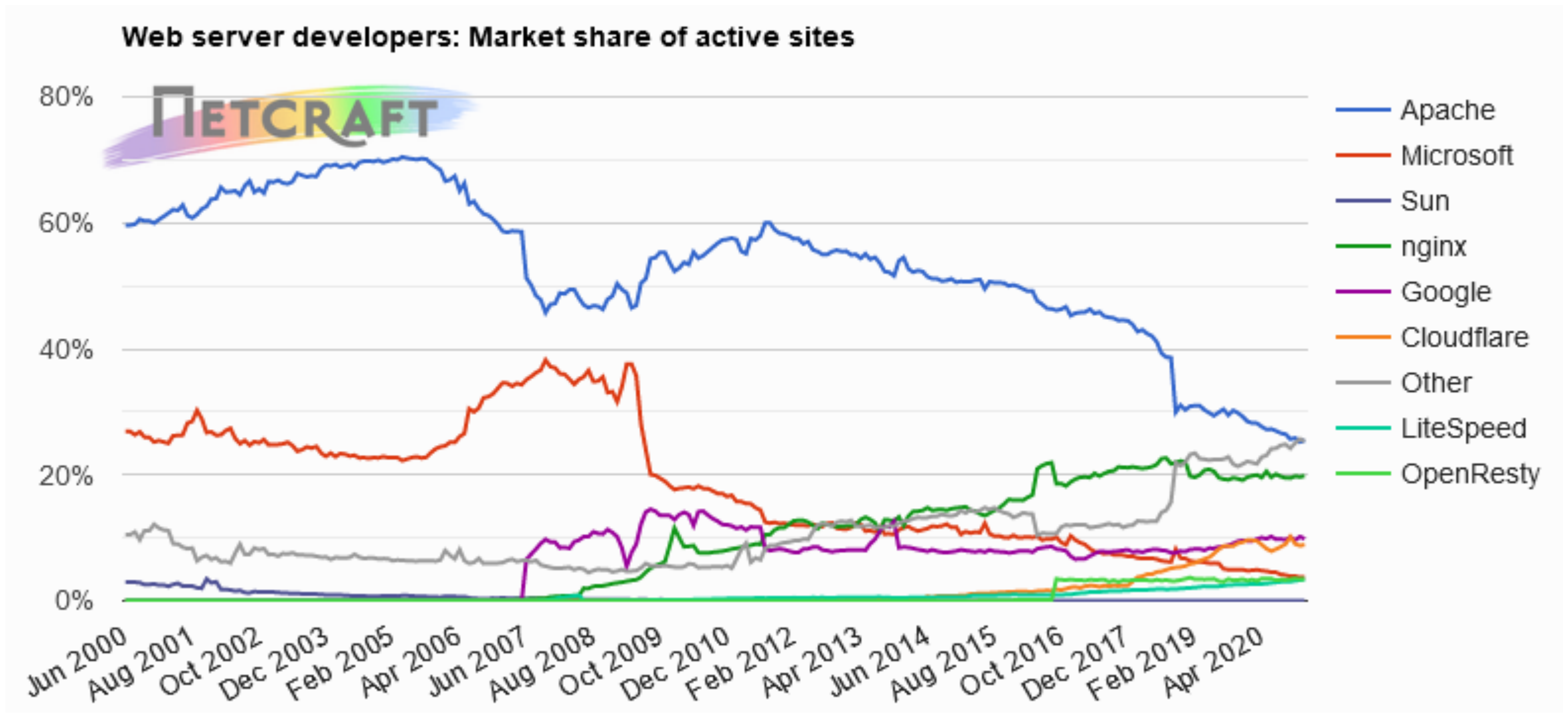
# Web Server



Source: http://news.netcraft.com/

https://news.netcraft.com/archives/2021/02/26/february-2021-web-server-survey.html

## Web Server



Source: http://news.netcraft.com/

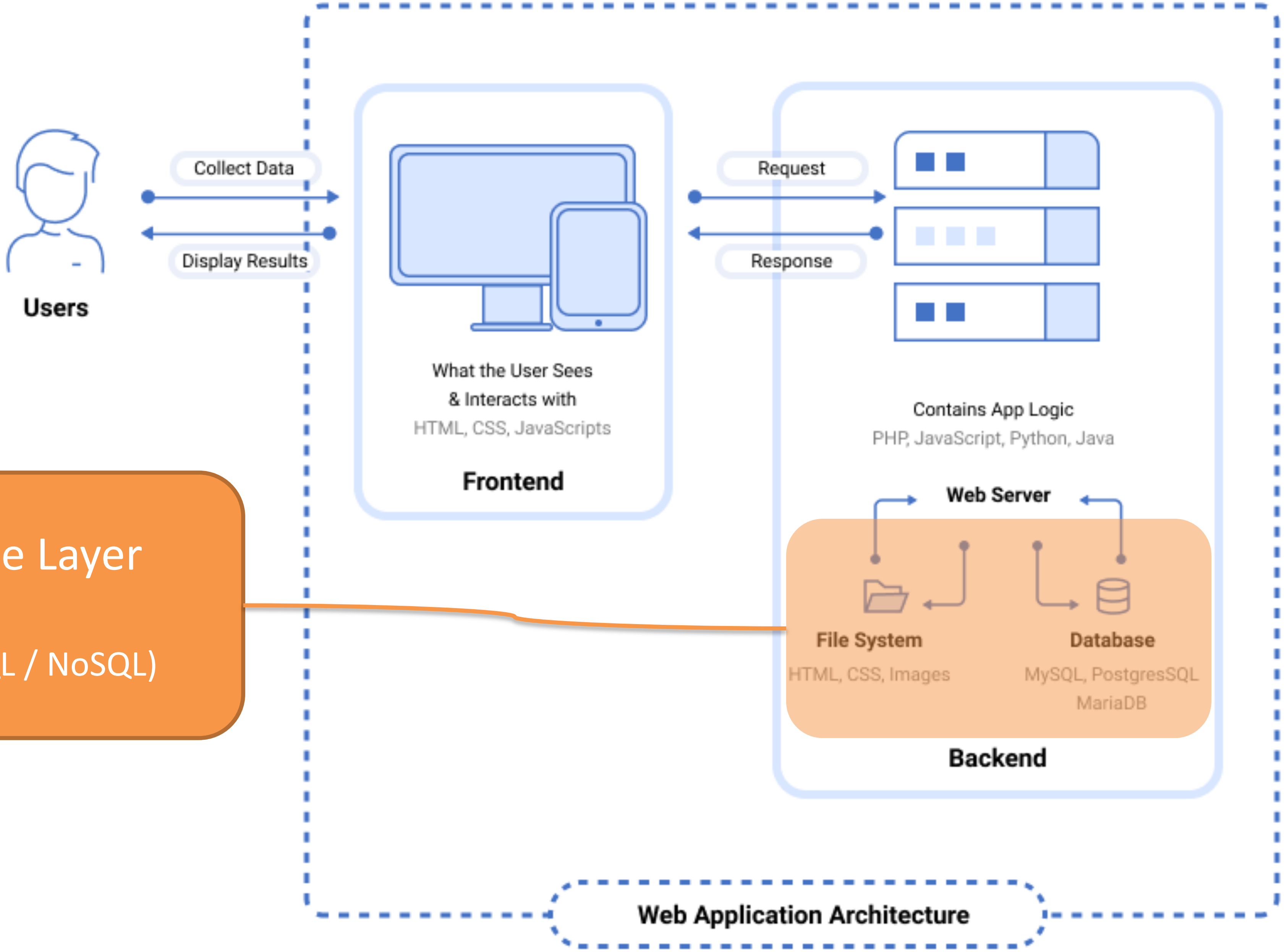
https://news.netcraft.com/archives/2021/02/26/february-2021-web-server-survey.html

# Web Server with Node.js

- Node.js provides a module ‘`http`’ that implements a basic web server
- `Express`: a simple and extensible web server, easy to extend with many available extensions - http://expressjs.com/
- Other alternatives:
  - Fastify: focuses on performance
  - Koa: by Express authors, simplifies callbacks using ‘ES6 generators’ (yield instruction)
  - Meteor: full-stack, more complex and complete, also with a client-side component to synchronize state
  - Sails.js: based on MVC+ORM principles
  - … many more

Persistence Layer

Databases (SQL / NoSQL)



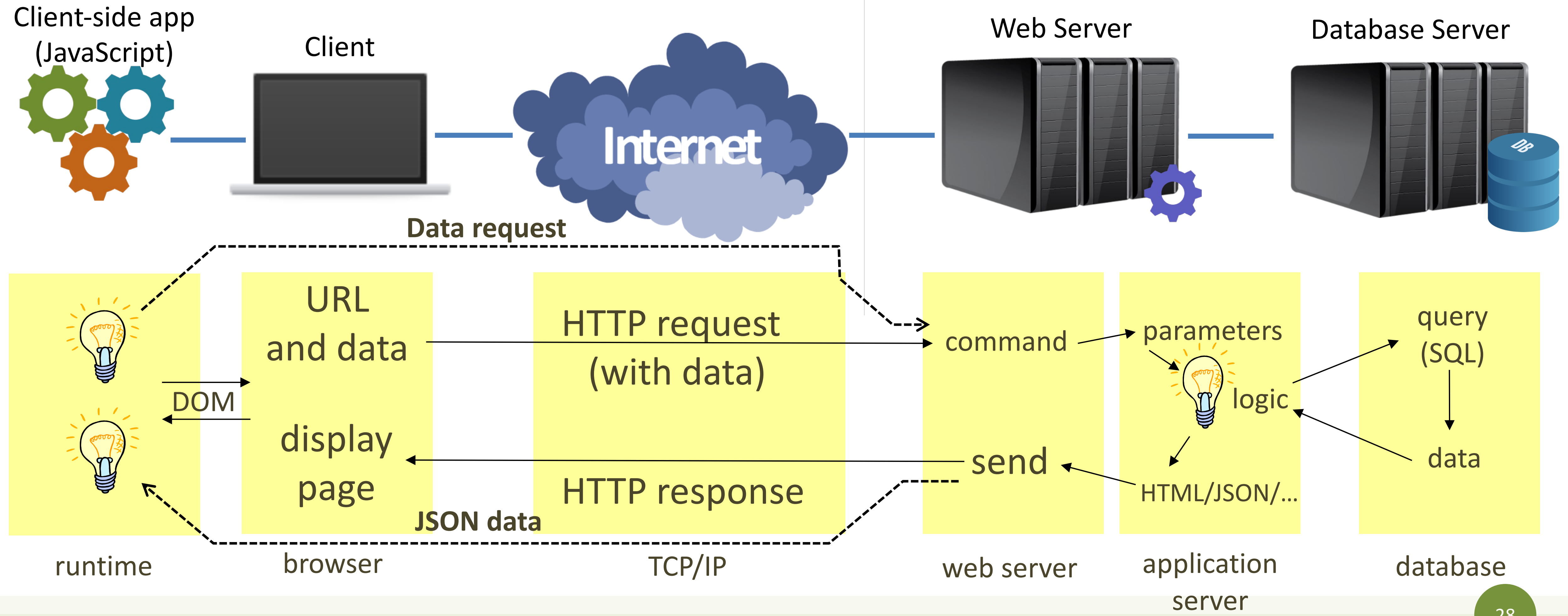
Web Architecture

## ARCHITECTURAL PATTERNS

## "Traditional" Architectural Pattern

- The "Rich-Client" is the "traditional" approach, now
- The server sends a new HTML page for each request it receives
  - with related resources (i.e., images, CSS, …)
  - some parts of those pages can be, then, dynamically updated with asynchronous JavaScript requests
- A web application is doing **server-side rendering**, and a *multi-page* web application is created

# All The Layers At Work...



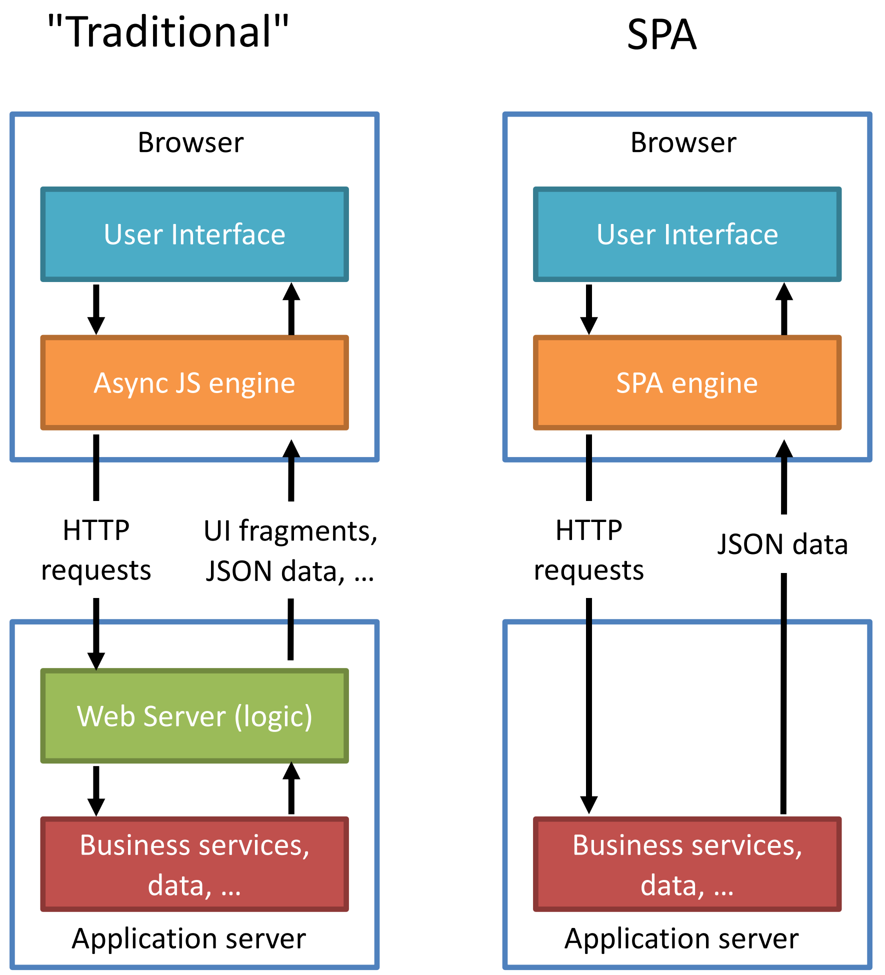
## Modern Patterns

Other three patterns to architect a web application exist, roughly

1. **Single-Page Application** (SPA)
   - the server sends the exact same web page for every unique URL
   - the page runs JavaScript to change the content and the aspect
   - by querying another (logical) server which provides "raw" information

# Single-Page Application

- An *evolution* of the "traditional" approach
  - JavaScript starts with an (almost empty) HTML
  - add all the content dynamically
  - instead of asking for data to update some parts of a well-formed page
- Goal: to serve an outstanding User Experience with no page reloading and no extra time waiting
- Examples: Google Docs, Trello



## SPA: Disadvantages

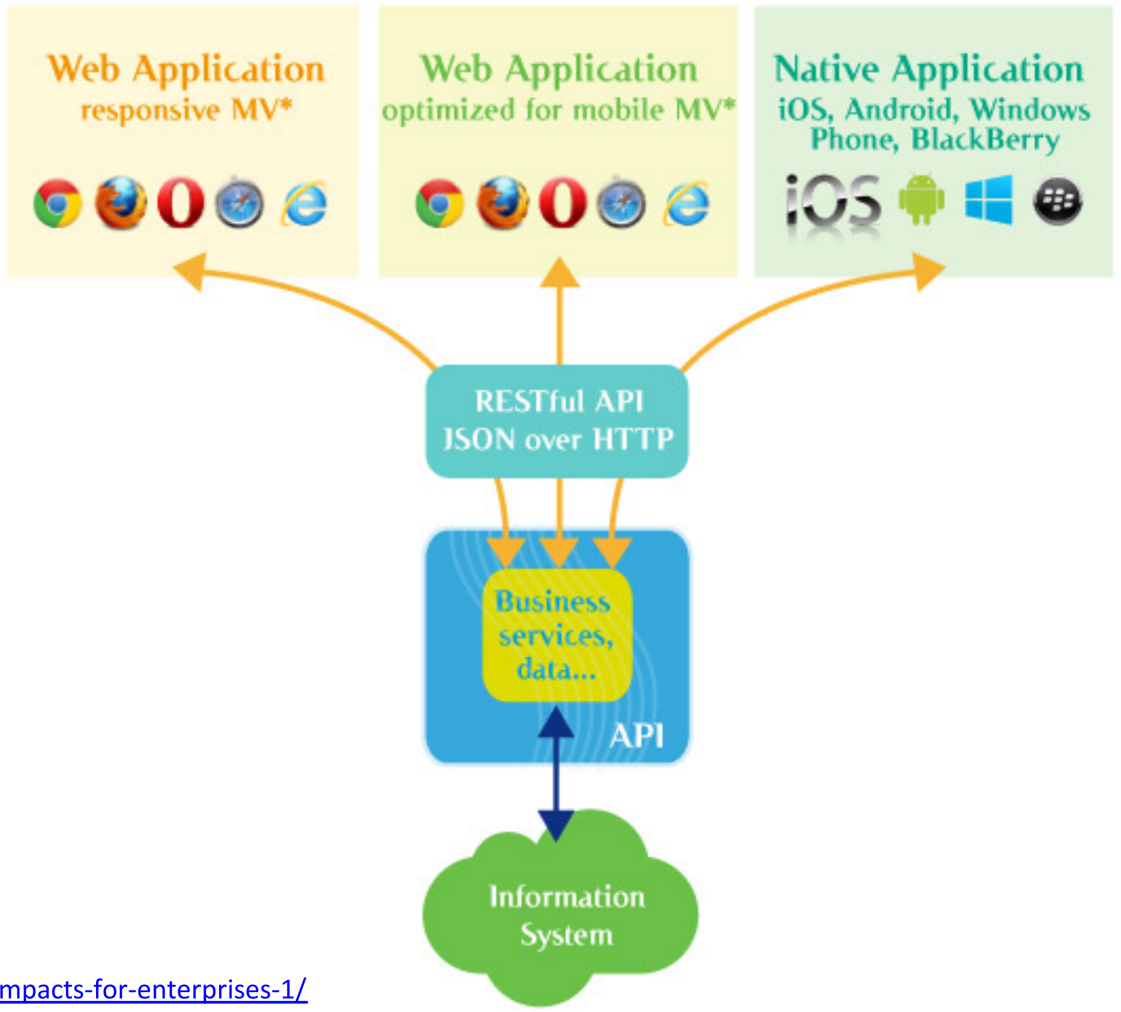
- Search Engine Optimization (SEO) is hard
  - Google launched a new scheme to increase single-page app SEO optimization, but this means extra work for the developer
- Browser history is not working
  - Web History API exists to tackle this problem and to allow a developer to emulate the back and forth action
- Security issues
  - Given that "all the logic is in the client", special care should be taken when handling access control. Cross-Site Scripting (XSS) is a problem as well.
- Client-side rendering can be slow!

# Modern Patterns

Other three patterns to architect a web application exist, roughly

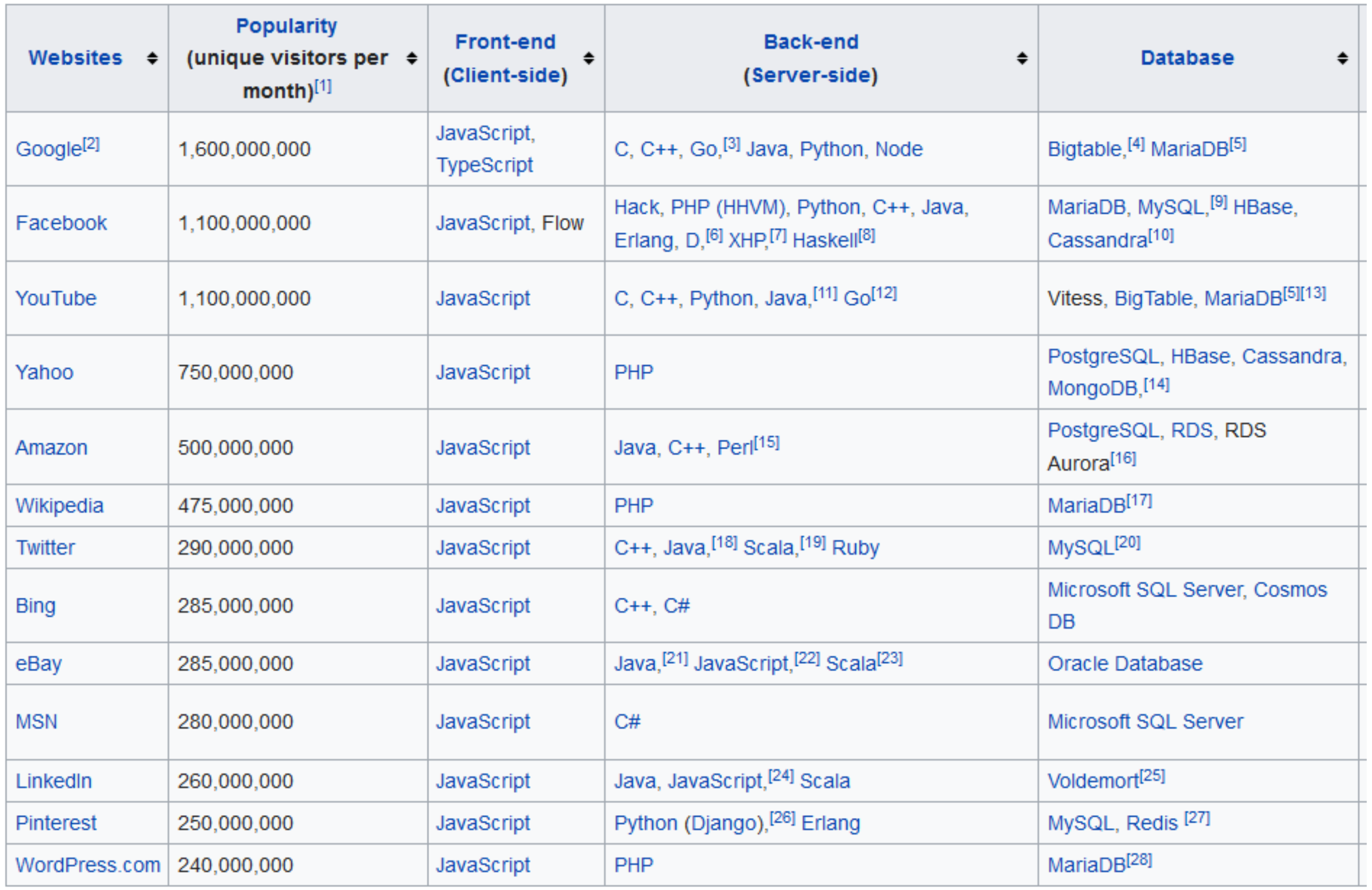
1. **Single-Page Application** (SPA)
   - the server sends the exact same web page for every unique URL
   - the page runs JavaScript to change the content and the aspect
   - by querying another (logical) server which provides "raw" information
2. **Isomorphic Application**
   - Combination of SPA with server-side rendering
3. **Progressive Web App (PWA)**
   - Web applications that emulate "native" apps

# Supporting mobile development



http://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/

# Client-side, server-side, databases



| Websites | Popularity (unique visitors per month)[1] | Front-end (Client-side) | Back-end (Server-side) | Database |
|---|---|---|---|---|
| Google[2] | 1,600,000,000 | JavaScript, TypeScript | C, C++, Go,[3] Java, Python, Node | Bigtable,[4] MariaDB[5] |
| Facebook | 1,100,000,000 | JavaScript, Flow | Hack, PHP (HHVM), Python, C++, Java, Erlang, D,[6] XHP,[7] Haskell[8] | MariaDB, MySQL,[9] HBase, Cassandra[10] |
| YouTube | 1,100,000,000 | JavaScript | C, C++, Python, Java,[11] Go[12] | Vitess, BigTable, MariaDB[5][13] |
| Yahoo | 750,000,000 | JavaScript | PHP | PostgreSQL, HBase, Cassandra, MongoDB,[14] |
| Amazon | 500,000,000 | JavaScript | Java, C++, Perl[15] | PostgreSQL, RDS, RDS Aurora[16] |
| Wikipedia | 475,000,000 | JavaScript | PHP | MariaDB[17] |
| Twitter | 290,000,000 | JavaScript | C++, Java,[18] Scala,[19] Ruby | MySQL[20] |
| Bing | 285,000,000 | JavaScript | C++, C# | Microsoft SQL Server, Cosmos DB |
| eBay | 285,000,000 | JavaScript | Java,[21] JavaScript,[22] Scala[23] | Oracle Database |
| MSN | 280,000,000 | JavaScript | C# | Microsoft SQL Server |
| LinkedIn | 260,000,000 | JavaScript | Java, JavaScript,[24] Scala | Voldemort[25] |
| Pinterest | 250,000,000 | JavaScript | Python (Django),[26] Erlang | MySQL, Redis [27] |
| WordPress.com | 240,000,000 | JavaScript | PHP | MariaDB[28] |

https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

# References

- HTTP/1.x vs. HTTP/2 – The Difference Between the Two Protocols Explained - https://cheapsslsecurity.com/p/http2-vs-http1/
- How Browsers Work: Behind the scenes of modern web browsers - https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
- Inside look at modern web browser
  - Part 1: https://developers.google.com/web/updates/2018/09/inside-browser-part1
  - Part 2: https://developers.google.com/web/updates/2018/09/inside-browser-part2
  - Part 3: https://developers.google.com/web/updates/2018/09/inside-browser-part3
  - Part 4: https://developers.google.com/web/updates/2018/09/inside-browser-part4

# License