INTRO APPWEB 2023

Web Architectures

Layer, Languages, Protocols

Luigi De Russis

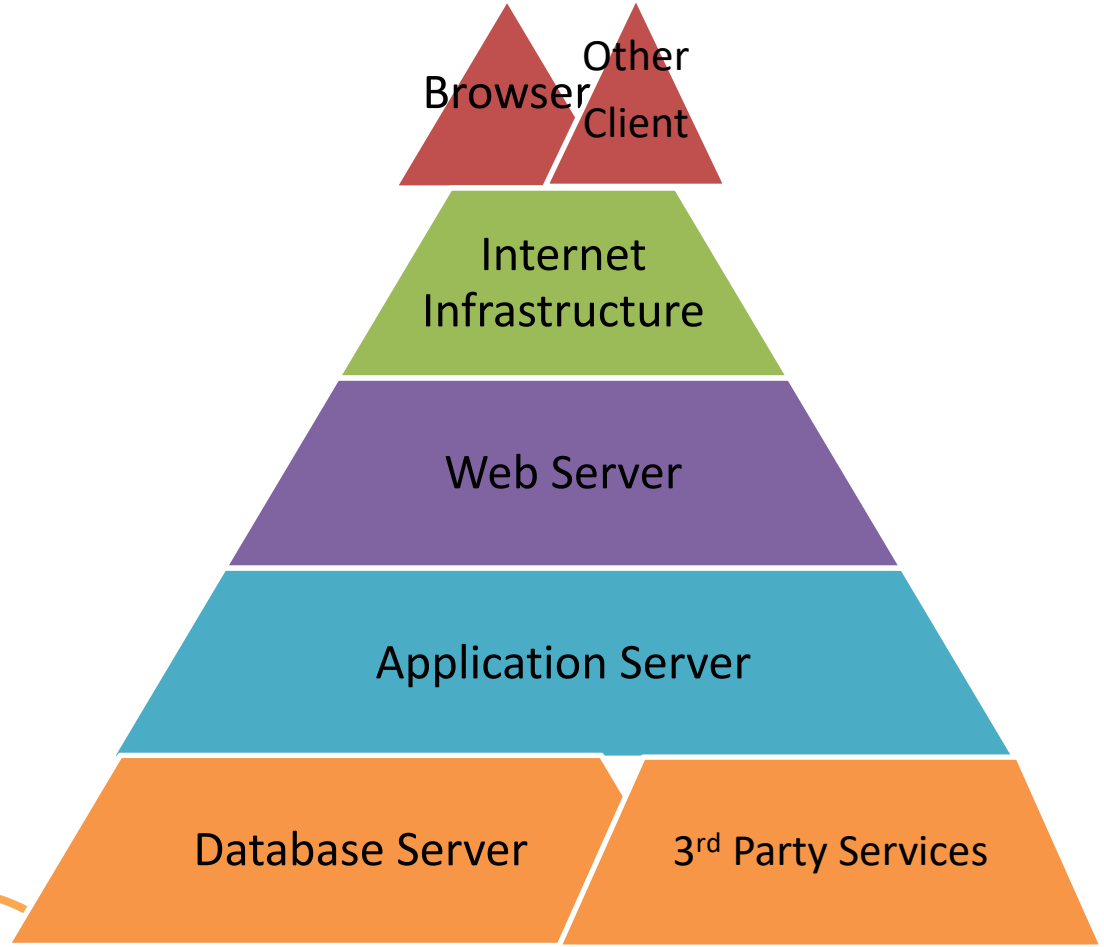
Politecnico di Torino

Goal

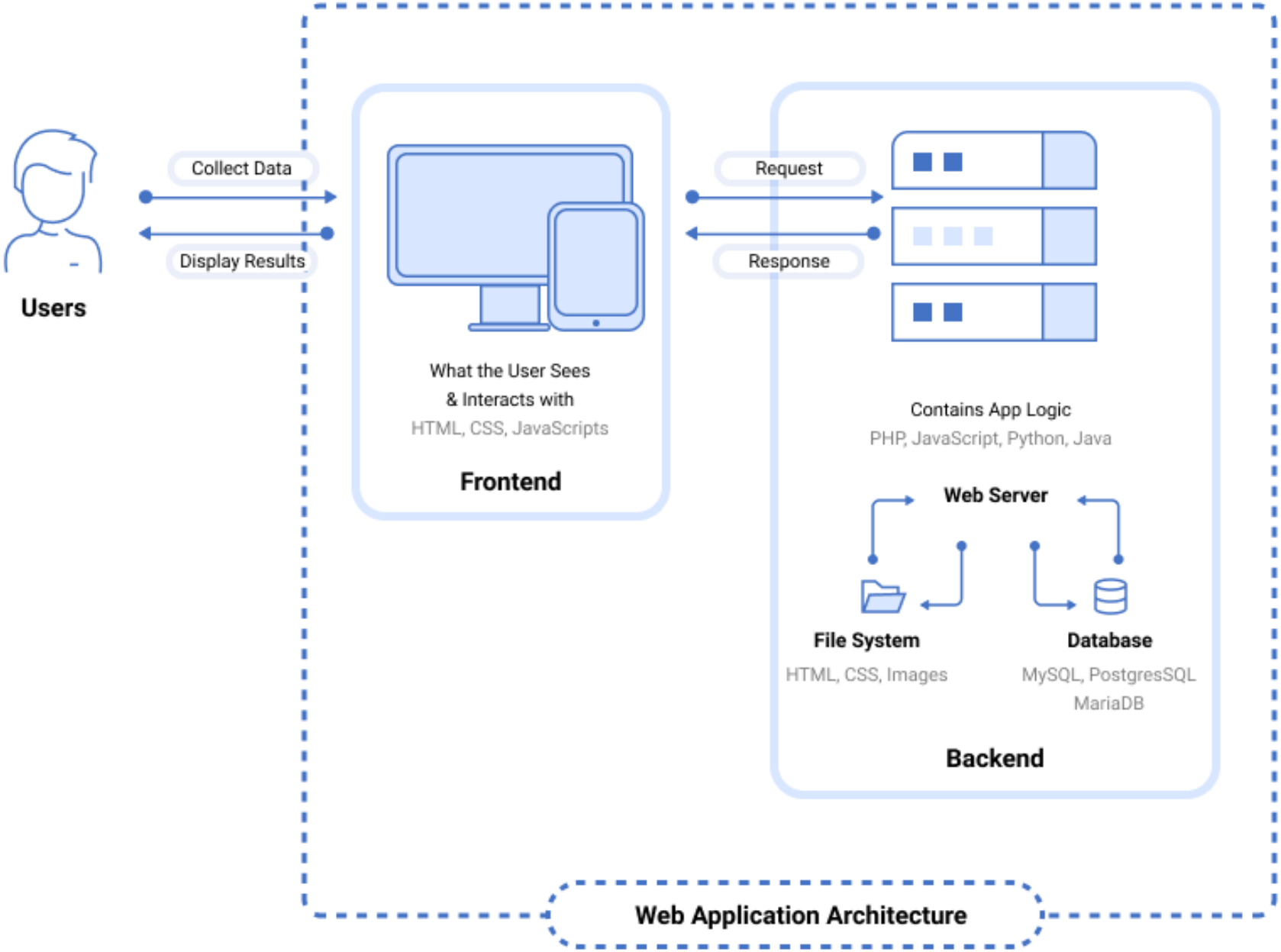
- Understand what is the Web and its architecture
  - main (logical) components
  - main network protocols
  - existing architectural patterns and languages
- Know the interaction and communication across components
- Learn the basics of how a browser works

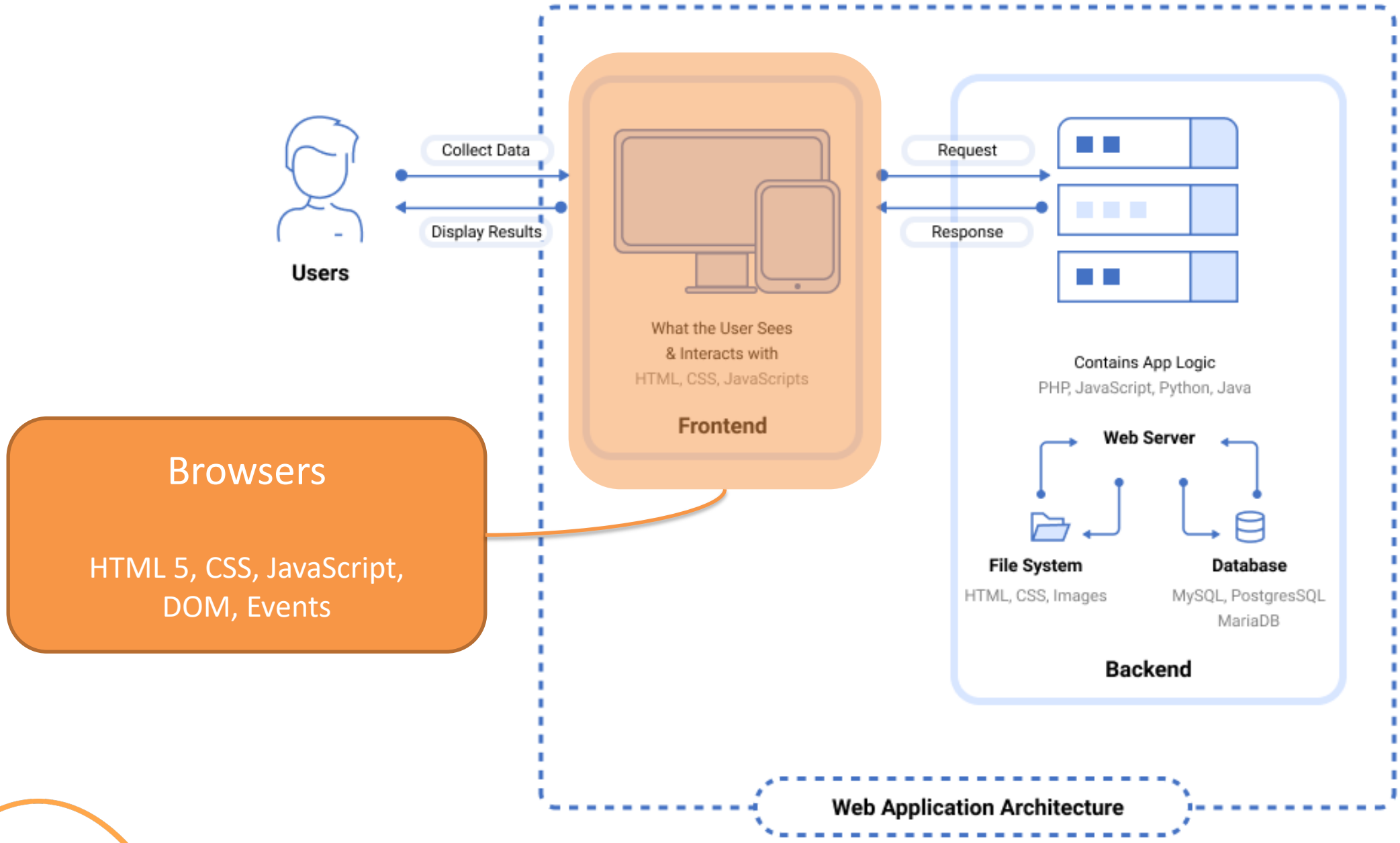
- *NOTE: Several of the topics mentioned here will be presented in more details along the course*

N-tier (N-level) Architecture



- Each level/tier has a well-defined role
- One or more servers implement each tier/layer
- More servers can share the same hardware or can run on dedicated devices
- Communication between tiers/levels is achieved through the network

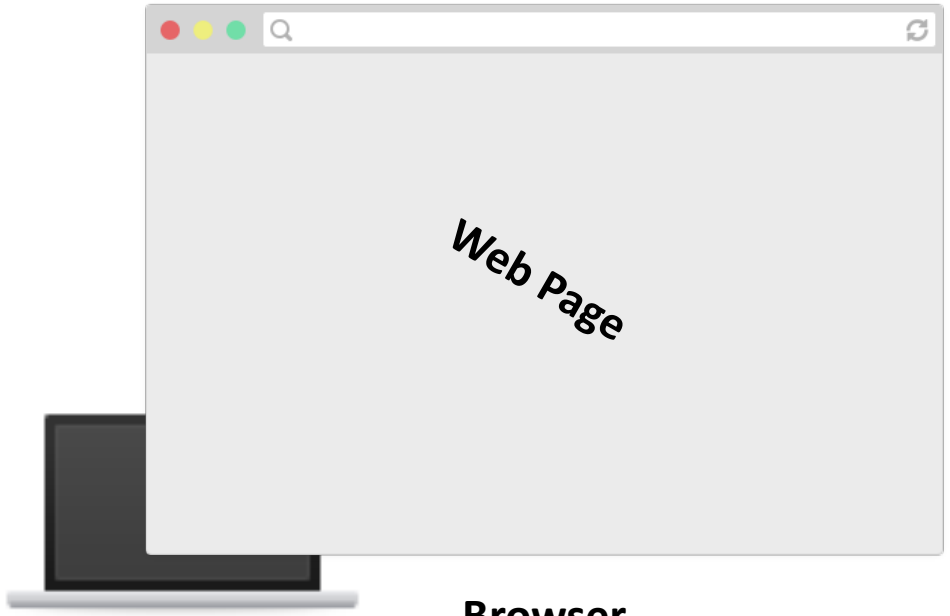

Users
Collect Data
Display Results
What the User Sees
& Interacts with
HTML, CSS, JavaScripts
Frontend
Request
Response
Contains App Logic
PHP, JavaScript, Python, Java
Web Server
File System
HTML, CSS, Images
Database
MySQL, PostgresSQL
MariaDB
Backend
Web Application Architecture



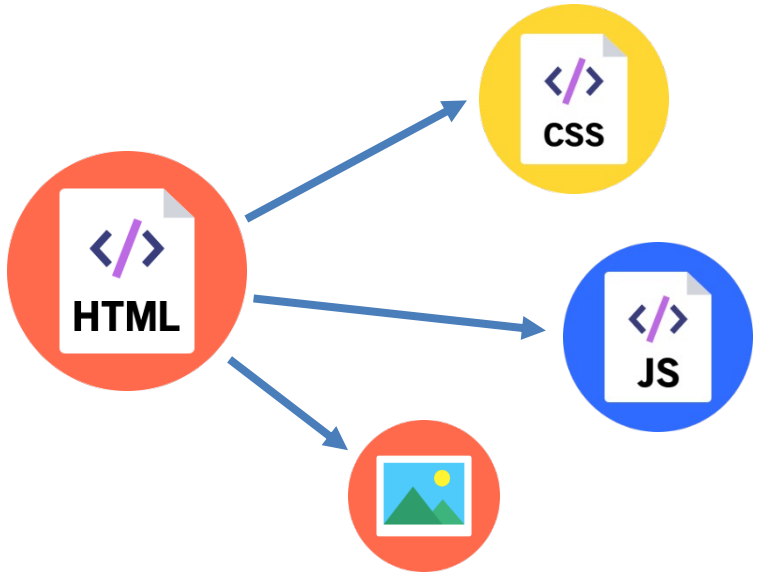
Browsers

HTML 5, CSS, JavaScript,
DOM, Events

Browser



Browser



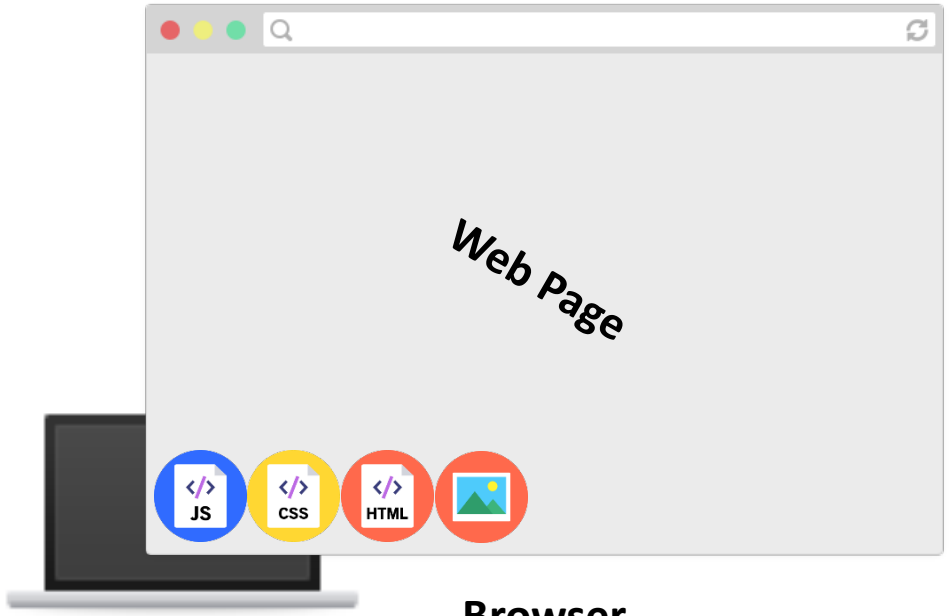
The HTML file might link to other **resources** (images, videos, …)
as well as **JavaScript** and **CSS** files,
which the browser then also loads

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

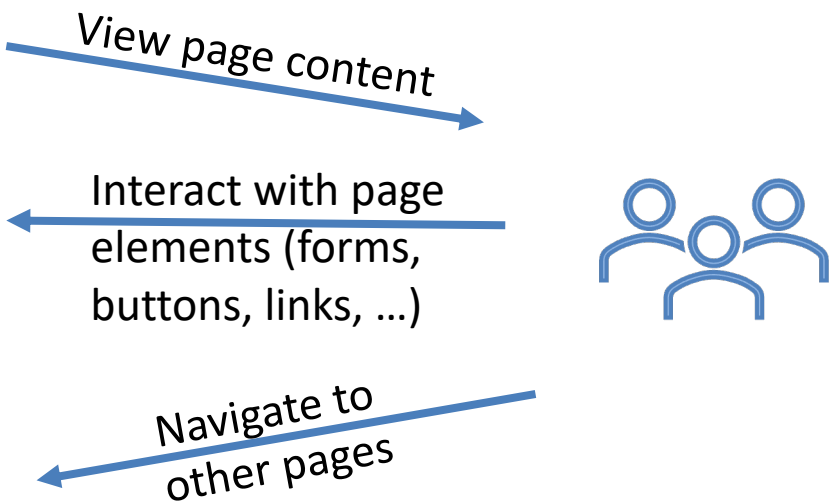
HTML

- Hyper Text Markup Language
- Defines the structure of a web page
- A series of “tags” with an associated *semantic* meaning
  - <html>…</html>
  - <body>…</body>
  - <header>…</header>
  - <img />
  - …

Browser



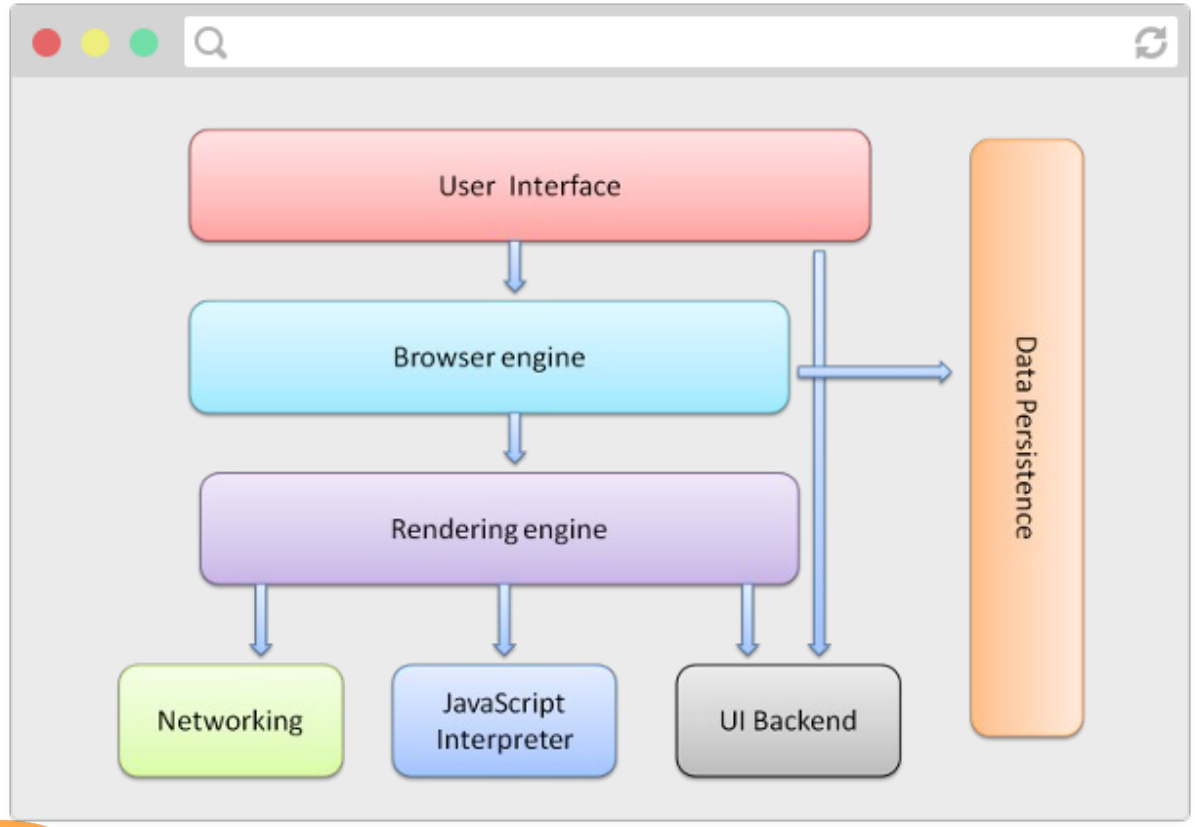
Browser



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

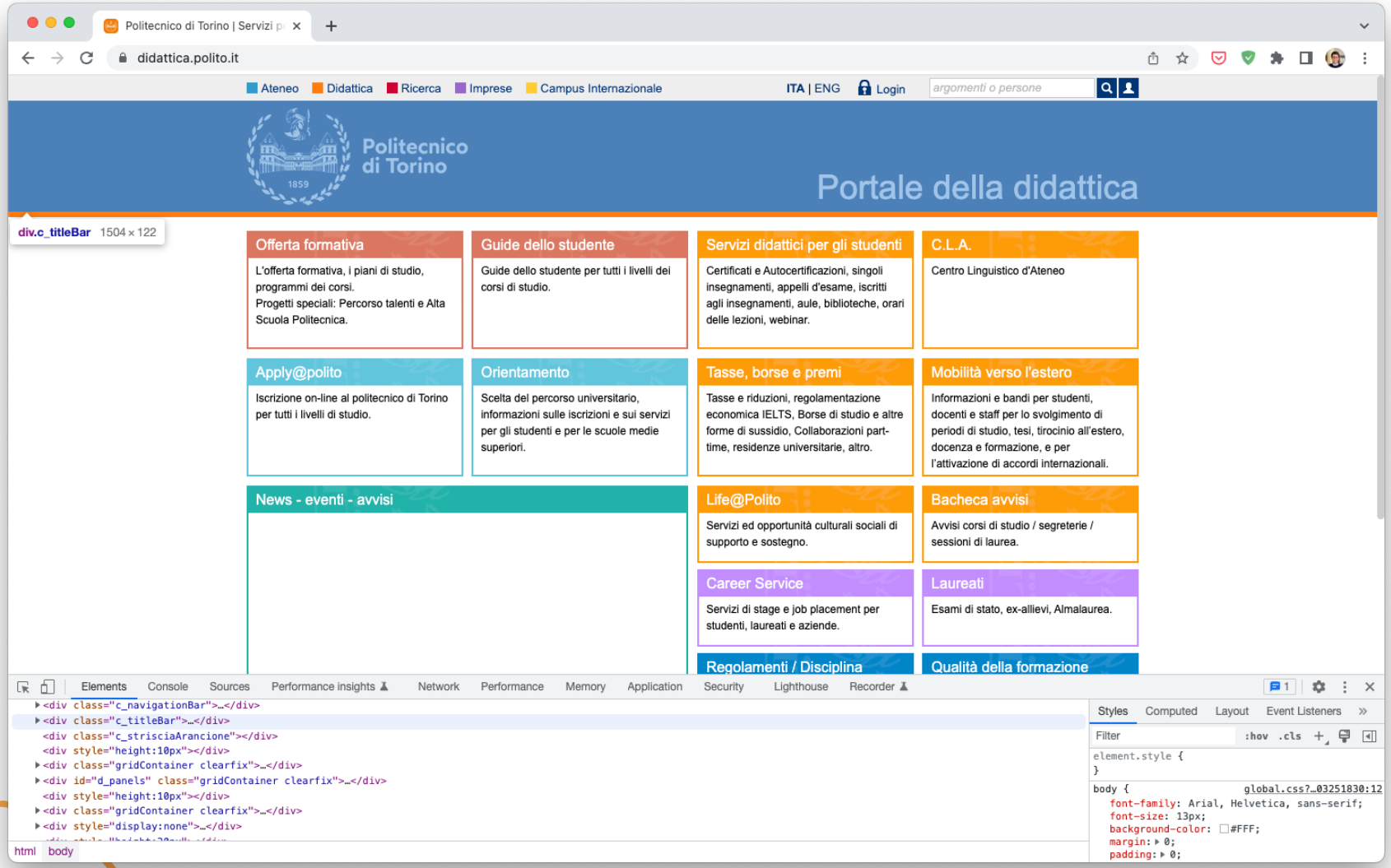
Clicking on a link brings the user to a **new page**.
Interacting with other elements may generate ***Events*** inside the browser.
Such Events are “captured” by JavaScript and may **update the page content**.

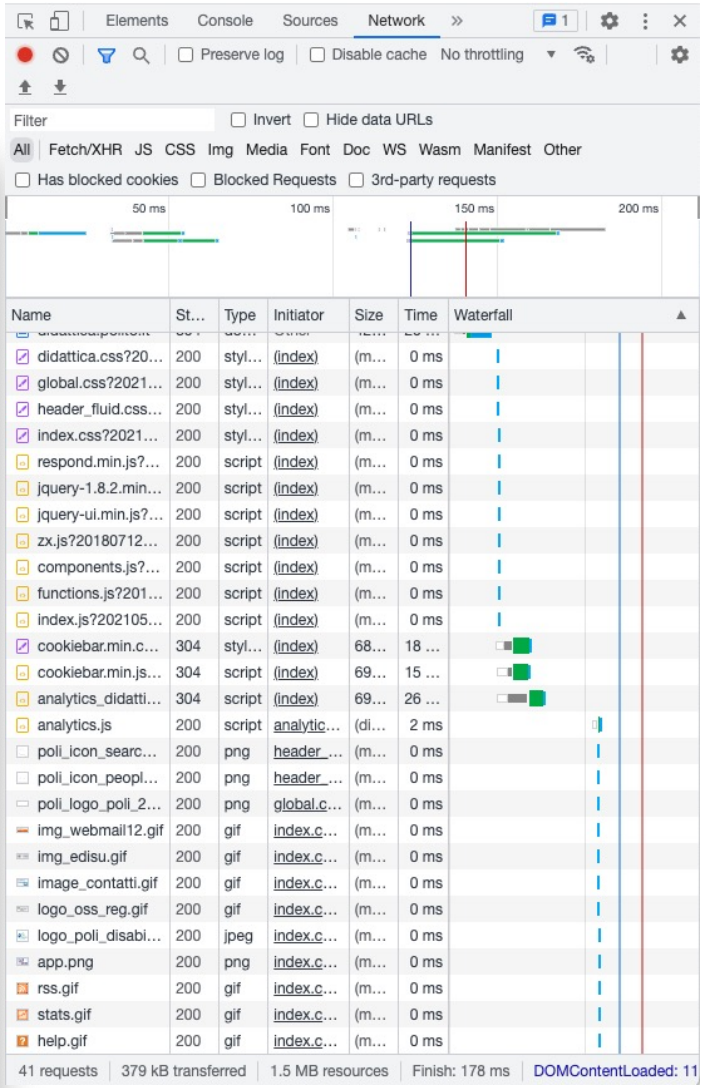
Conceptual Browser Architecture (from 10,000 feet)



- **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

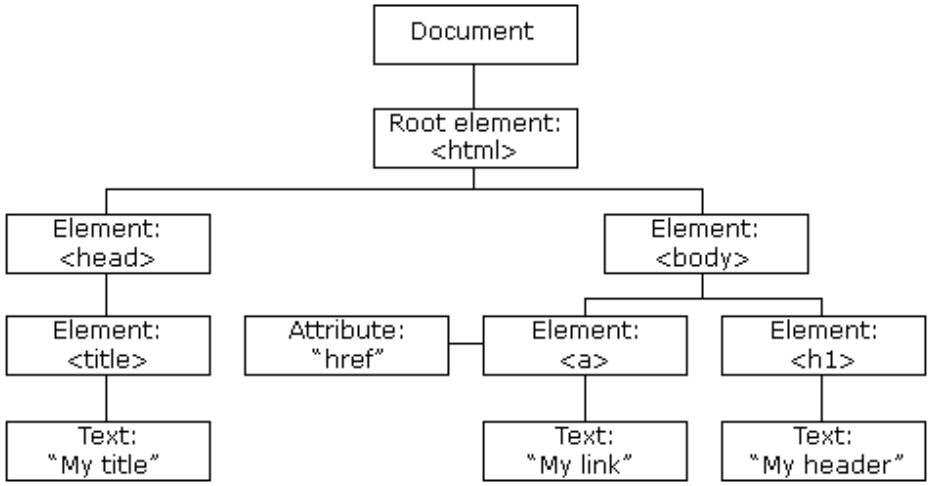




Document Object Model (DOM)

"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

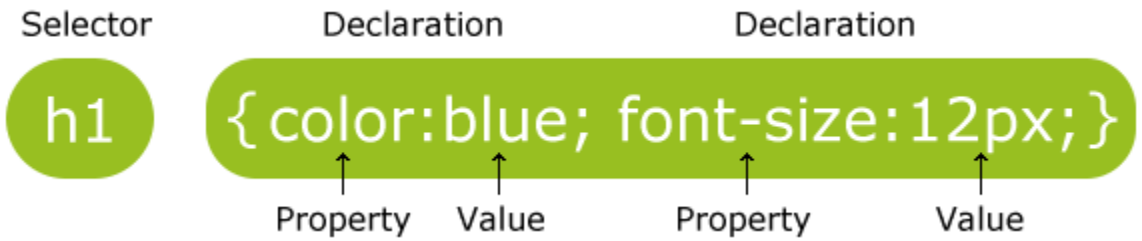


Cascading Style Sheets (CSS)

- Define the style and appearance of a web page
- 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

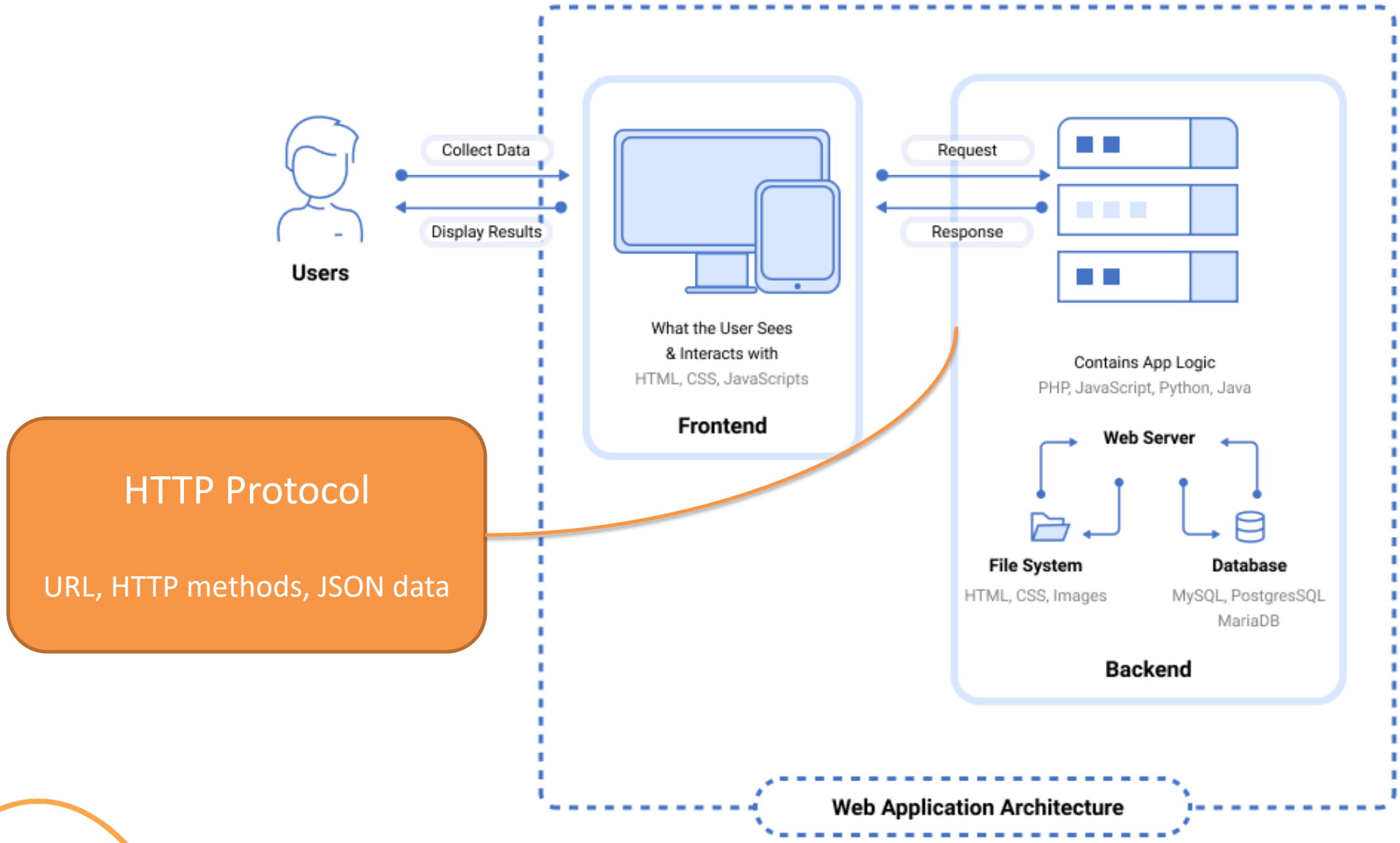
Cascading Style Sheets (CSS)

- 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

URL, HTTP methods, JSON data

Uniform Resource Locator (URL)

- 

http://www.sadev.co.za/users/1/contact
- 

http://www.sadev.co.za?user=1&action=contact
- 

https://bbd.co.za/index.html#about

HTTP Protocol

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

```
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 (Windo
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

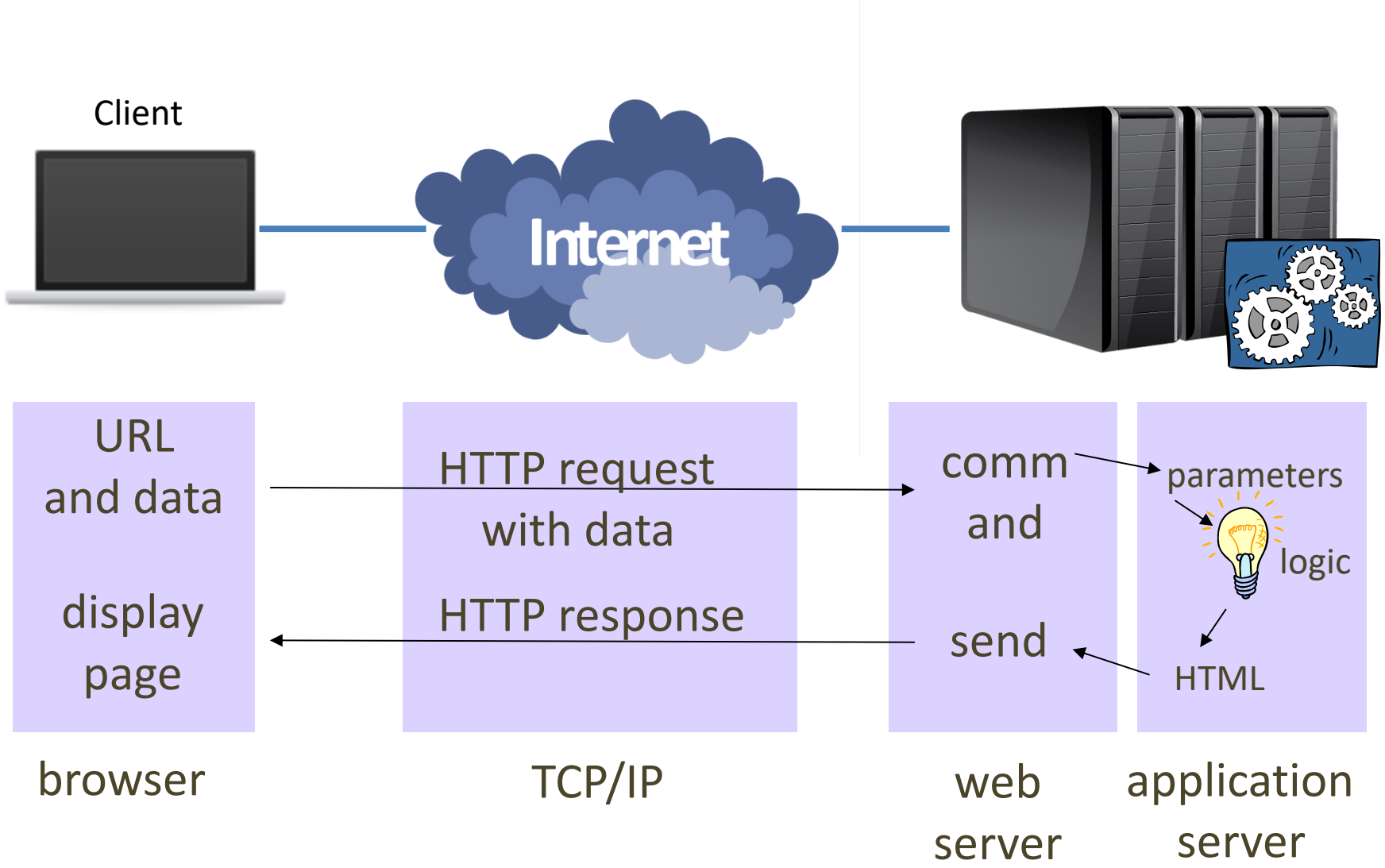
Generation

- 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

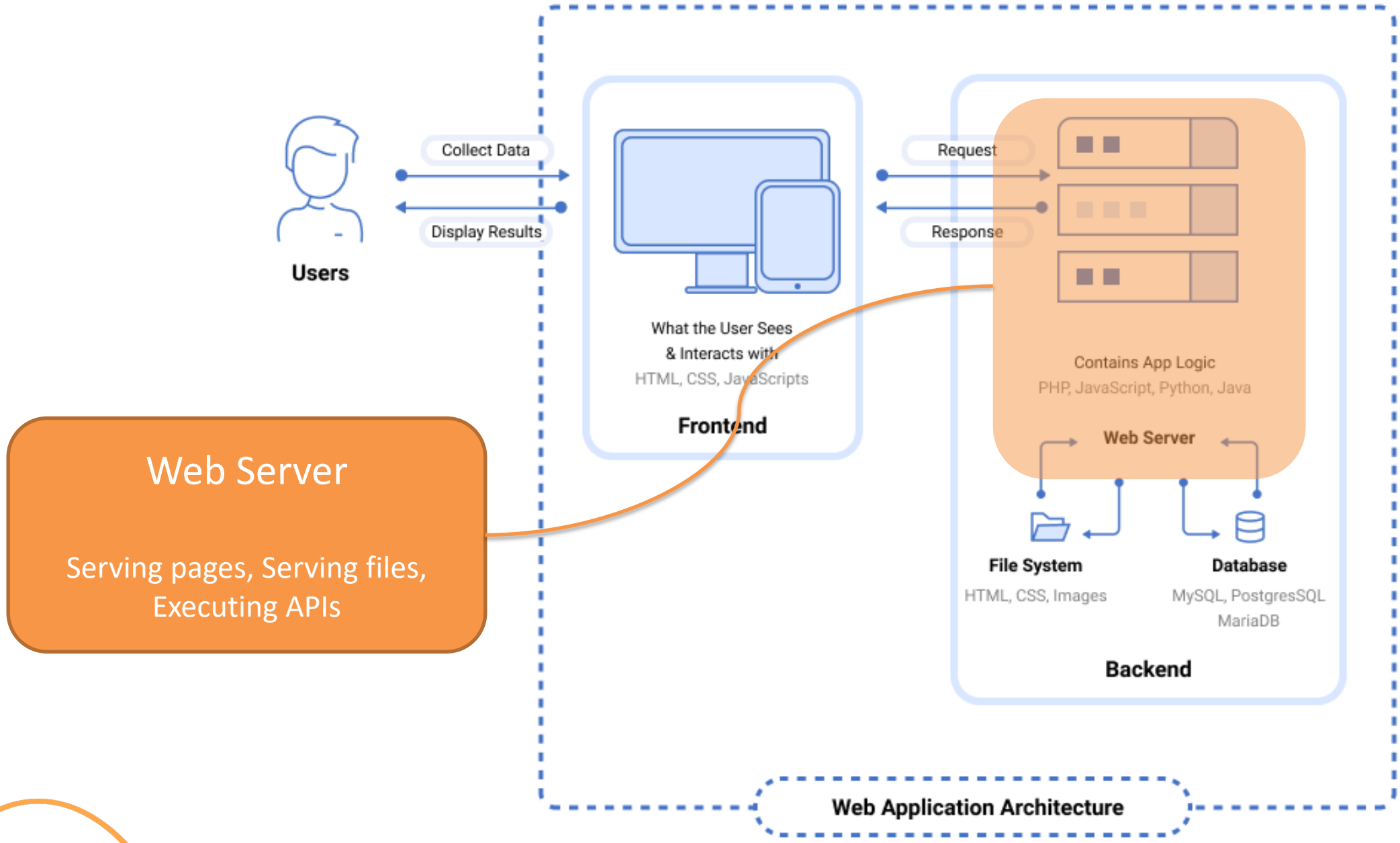
Dynamic Web Transaction



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 APIs

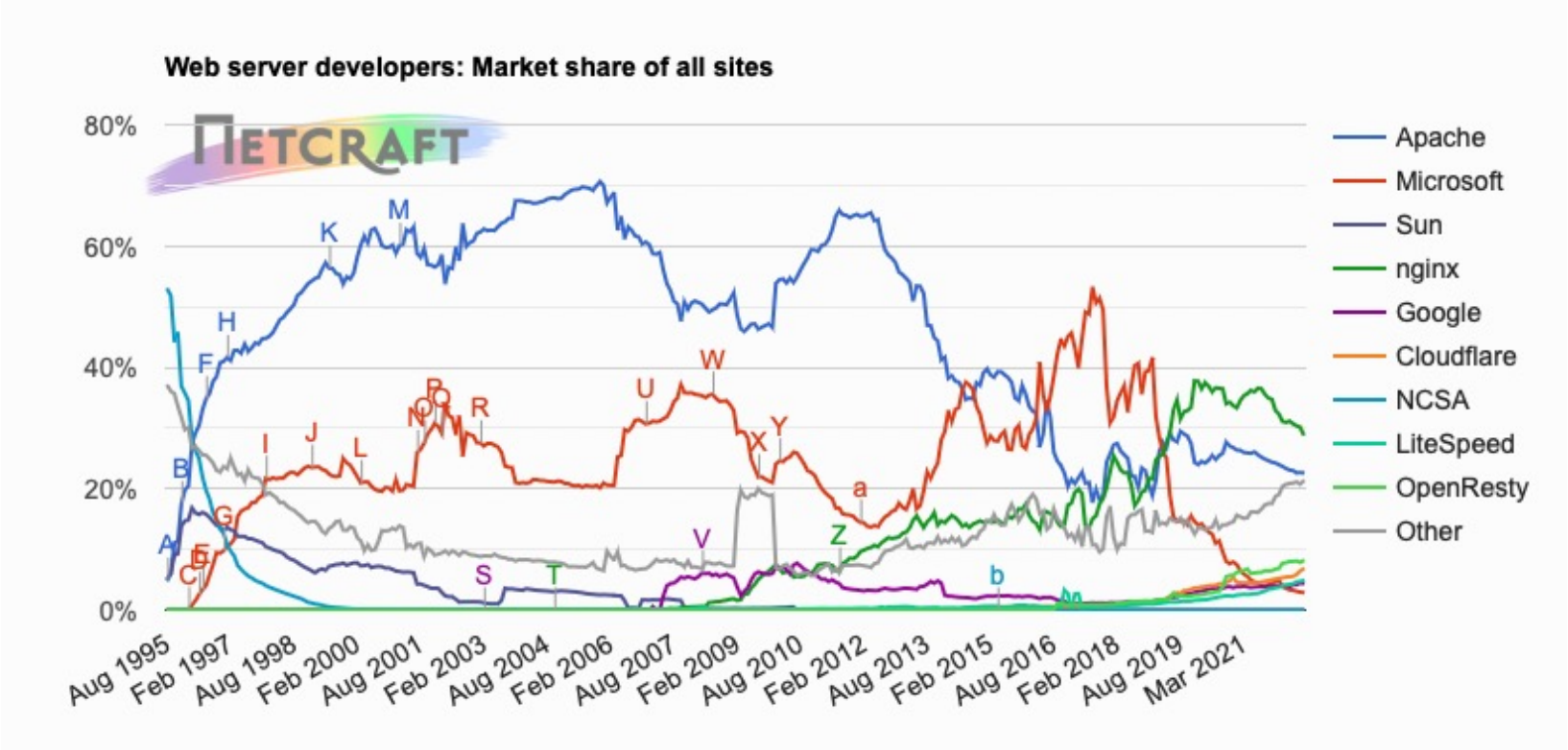
Server

- Logical definition
  - A process that runs on a host that relays information to a client upon the client sending it a request
- Physical definition
  - A host computer on a network that holds information (e.g., Web sites) and responds to requests for information

Web Server

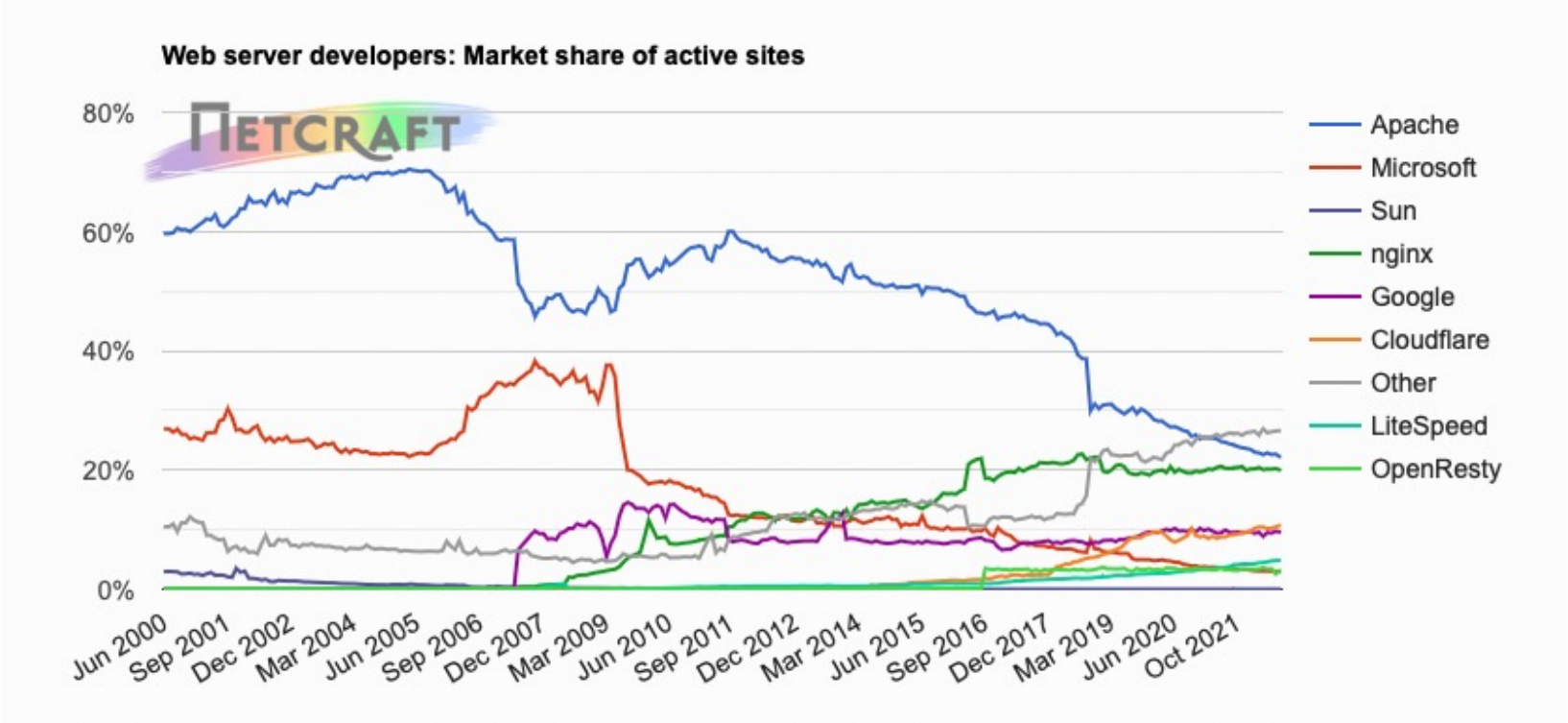
- A web server delivers web resources in response to a request
  - manages the HTTP protocol to handle requests and provide responses
- It either reads or generates a web page
  - receives client requests
  - reads *static page* from the filesystem
  - asks the application server to generate *dynamic 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 Servers… in the wild



source: https://news.netcraft.com/archives/2022/08/26/august-2022-web-server-survey.html

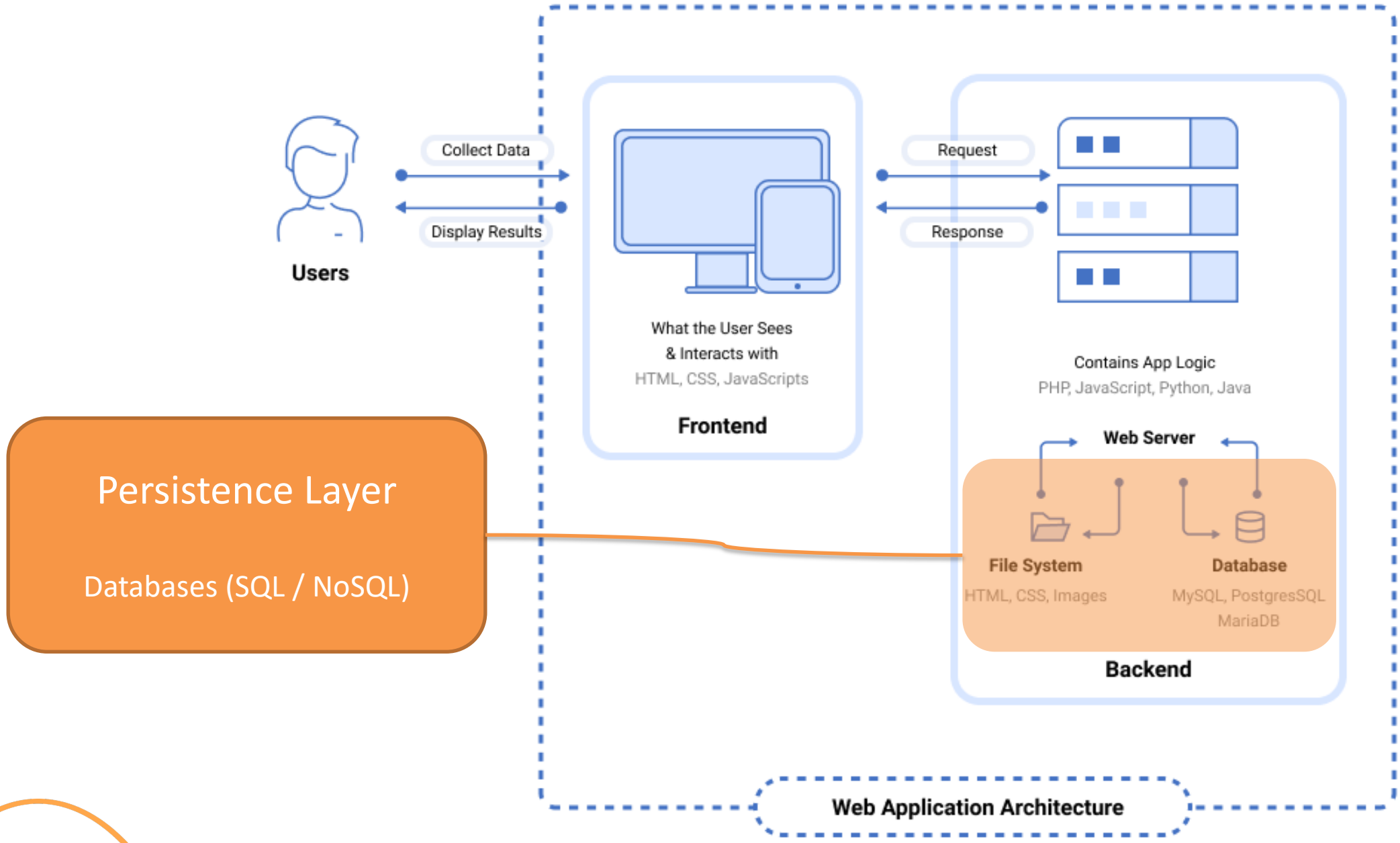
Web Servers… in the wild



source: https://news.netcraft.com/archives/2022/08/26/august-2022-web-server-survey.html

Web Server con Python

- Python provides a "`http.server`" module that implements a basic web server
- Flask: a simple web application framework, easy to extend with various available extensions
  - https://flask.palletsprojects.com/
- Other alternatives:
  - Django: very popular full-stack web framework
  - Tornado: focus on performance
  - Bottle: simple and fast micro-framework
  - web2py: full-stack web framework with various conventions
  - …



Persistence Layer

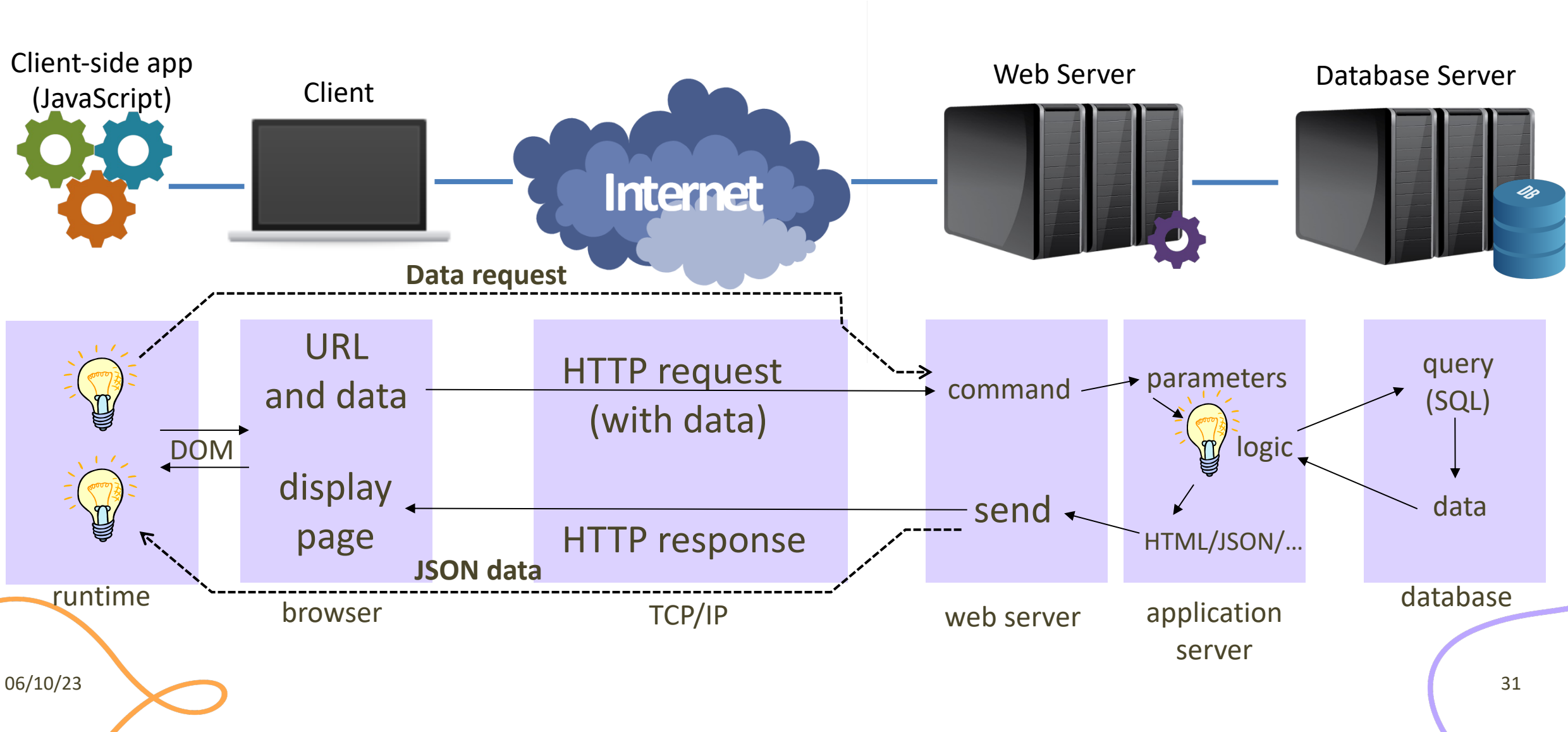
Databases (SQL / NoSQL)

ARCHITECTURAL PATTERNS

“Traditional” Architectural Pattern

- The so-called “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

Rich Client: All The Layers At Work…



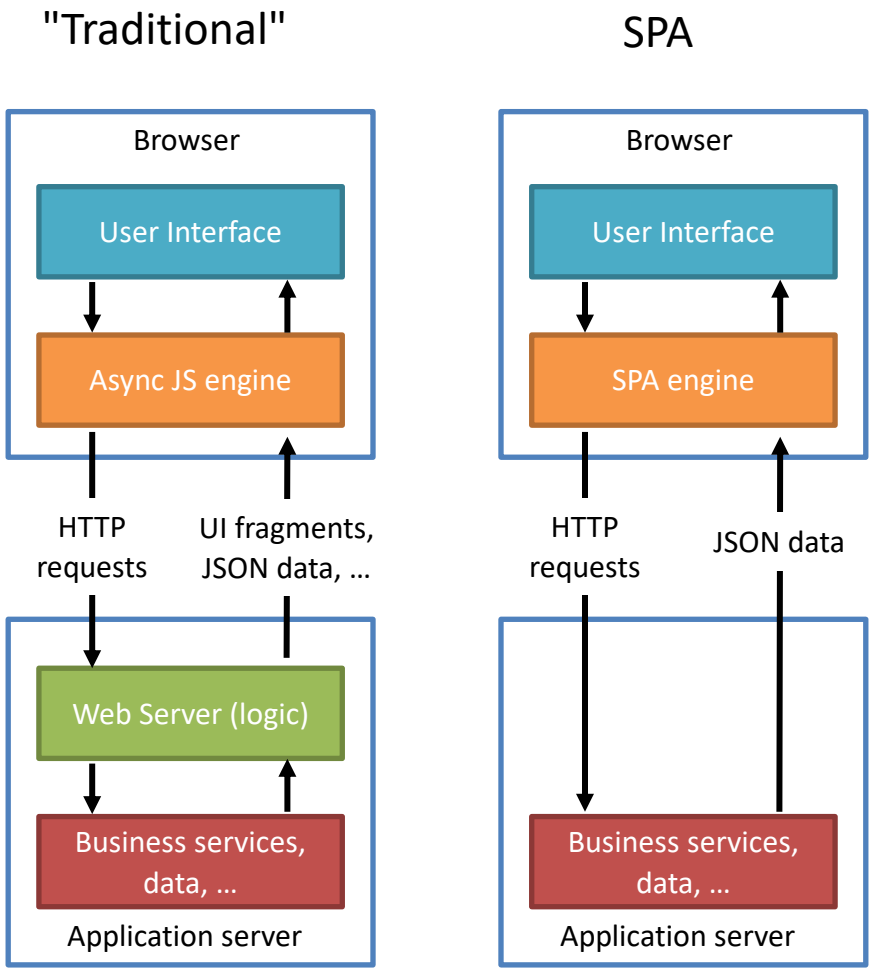
Modern Patterns

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 (SPA)

- 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

Front-ends, Back-ends, 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]

source: 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

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/