Lezione09-Web Architecture

Saturday, March 13, 2021 4:15 PM

<WA1/>
<AW1/>
2021

Web Architecture

Layers, Languages, Protocols

Fulvio Corno
Luigi De Russis
Enrico Masala

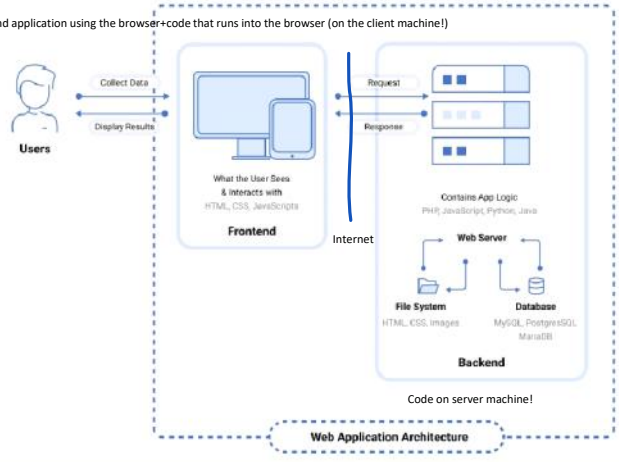
Goal

- 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

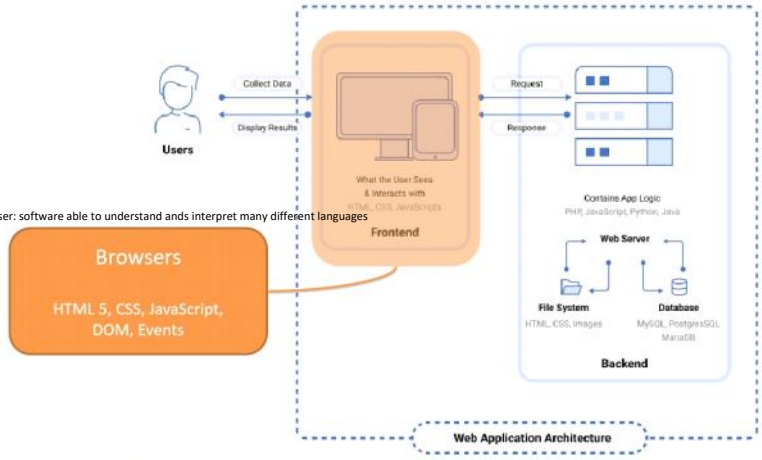
- *NOTE: All the topics mentioned here will be presented in more details in the next lectures*

Overview of components and architectures used today.

We offer a service.
User interacts with front-end application using the browser+code that runs into the browser (on the client machine!)



Code on server machine!



Browser: software able to understand ands interpret many different languages

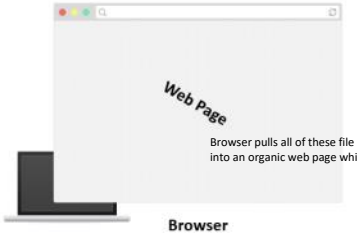
Browsers

HTML 5, CSS, JavaScript, DOM, Events

Browser

"tool(software) for displaying and interact with web pages"

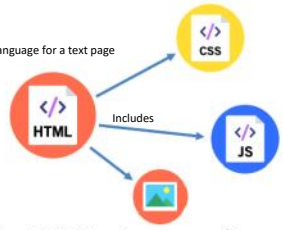
Cascade Styles Sheets: fonts, colors, spaces, alignments..



Browser

Browser pulls all of these file and compose them into an organic web page which is what we see

Simple descriptive language for a text page



Includes

Dynamic behaviour

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**

Ex. Images and js code are files always equal, stored on the server. Browser just needs to ask for them and server will send them!

Html files are not stored, because they are not always equal! There is an application software which will generate the html file! When browser asks for them, the server will first generate them on the fly, and then deliver them!

Quick Introduction to HTML

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

Browser

Objective of the browser

1 html, 0 or more images, css files, js files

View page content

interactive

Interact with page elements (forms, buttons, links, ...)

Navigate to other pages

This discards all of the previoulsy loaded files and states of the application, to load a new page FROM THE BEGINNING!
This is a big waste for us, so we'll try to create a single page application which magages navigation INTERNALLY, keeping the state! (faster and higher control?). (React Framework)

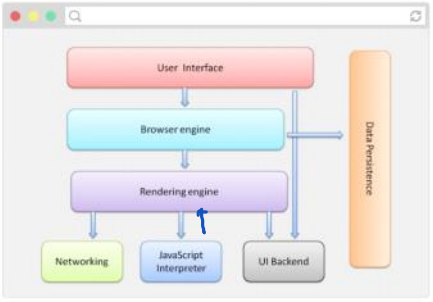
The content of the web page is described by HTML+CSS.

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**.

Events are handled by the browser itself automatically.
There is a predefined behaviour associated with the various actions, but our js code may also capture 1 or more of this events and customize what happens when they happen!
This is how we create interactive events! We capture user events and respond to them (define what to do).

Conceptual Browser Architecture (from 10,000 feet)

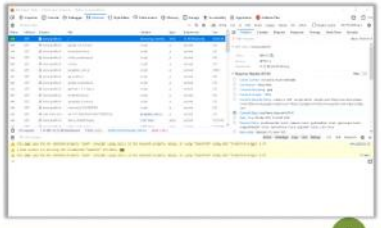
Today complex almost as much as OS! Both for functionality and efficience reason! Everything must be optimized!



- **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 — Catch user events interacting with os
- The **Browser Engine** marshals actions between the UI and the rendering engine — Links user behaviour with page content
- **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 — Most complex part. Renders and dispatch operations to networking module, js interpreter and ui backend.
- **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 — Render graphic
- **JavaScript Interpreter:** used to parse and execute JavaScript code — JS actions modifies page aspects
- **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 — Store info in a single page/ across different pages, with api accessible also to js

Browser Development tools

Html content, for each element which styles are applied to them and their measurements



Http requests that the browser is doing

Style editor to modify css

Js console which can access elements of the page

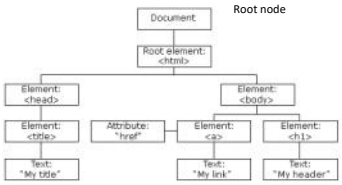
Debugger for js code (if not minimized, otherways it becomes difficult!)

What we modify is the current page displayed (not the application itself, because we are just on the client machine!) but once we found the bug, we can modify the server to serve the right pages!
Debugger almost better then the node.js debugger itself?!

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



Since the final page will be a combination of a text file (html), with styles and measurements applied to it and custom elements, it would be very difficult for our js code to work on the page only thinking about it in textual terms!
So the DOM was created. Provided by the Browser together with an API! This is shared between browser and js code. When JS code modifies something on the page, it is immediately displayed. When user interacts with the model (web page) in some ways, it is immediately displayed AND the js code immediately acknowledges it. (immediately=asynchronously)
Object programming model to represent a Document=Object-based data structure that describes the current content of the page as a tree. In this way the JS can use the API to modify the page! Way easier!

We need to understand: how the DOM represents the page and how does the API work (find, add, modify elements, manipulate DOM). Interact with the web page using objects

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

Styles changes how the html dom nodes are mapping into the pages (dimension, color, aligment..)
Smart! High level rules to compose the layout even with varying zoom level, orientation of the device, type of device and so on (not a fixed layout)

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, ...

Language based on rules that selects some elements and apply some properties to them!
All titles in the page will be of that color and that font size. Easy language to select elemets (similar to regex(pattern matching language: select elements), but many properties to understand (only difficult part). Today only modern design! (lower level design mechanism we don't see).

JavaScript

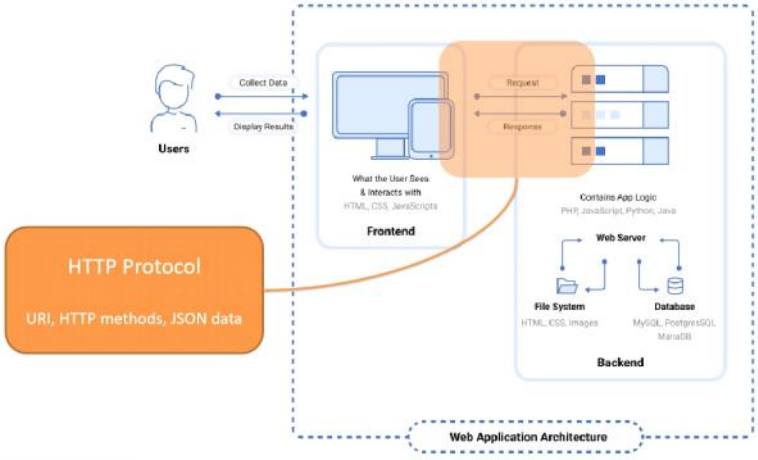
- JS Interpreter Embedded in the Browser
  - Executes within a strict "sandbox"
- JS Scripts loaded by the HTML page
  - `<script src="/js/myscript.js" type="text/javascript"></script>`
- JS Scripts have read-write access to
  - Browser API
  - HTML DOM (including form data)
  - User events and actions

N.B. Every time I open a page I execute someone else's code on my machine! A sandbox is required!! Every tab has its own sandbox (isolated env in which code is executed). (can't access client's files for example!)

Browser automatically loads scripts from server and runs it in the context of the page that requested it!

Provided by the server

Could control all control over the page or leave some of it to the browser



HTTP protocol

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

Command resource version

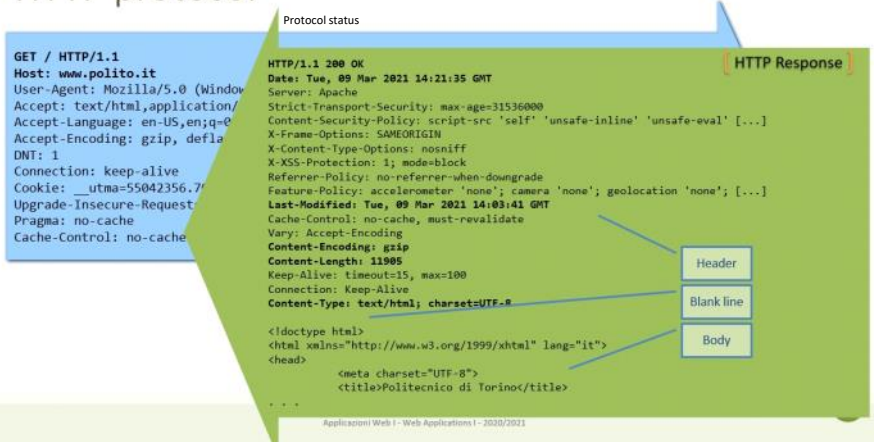
```
GET / HTTP/1.1
Host: www.polito.it
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive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

HTTP Request

Browser sends this to the server!
We can modify the various header from our js code!

HTTP protocol

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



Browser is always the one to ask requests to the server!

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

Files written by hand or generated ahead of time by a compilation before sending it!

Often 1 html (static) template for multiple pages, then filled depending on status

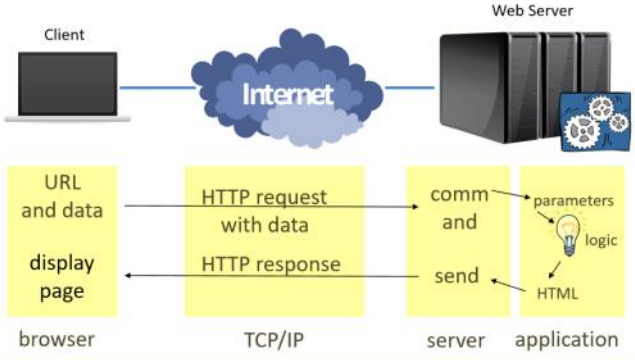
Personal data

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

JSON can easily be transformed into a JS object that we will manipulate to update page content

Dynamic Web Transaction



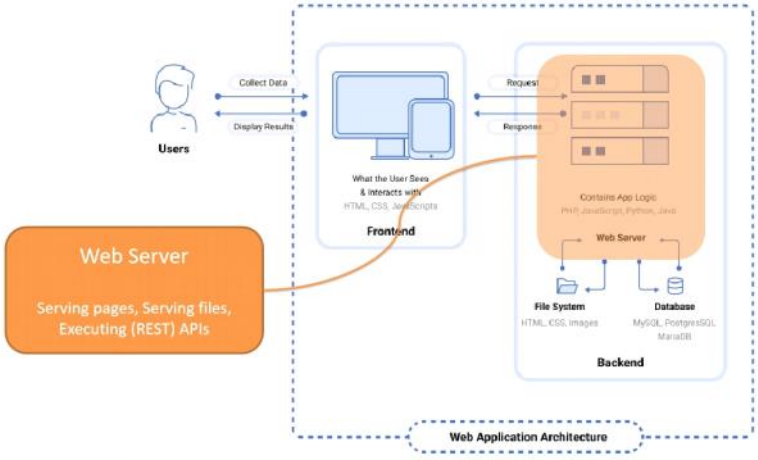
Server Side Application

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

Get, Post, Put, Delete
Can be used also to exchange data from the browser to the server both for showing the page and for our js code to request resources from the server



Receives requests and responds to it directly (static responses can be done automatically accessing servers file system) or using a programming language to generate the requested page and return it!
We'll use Express.

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

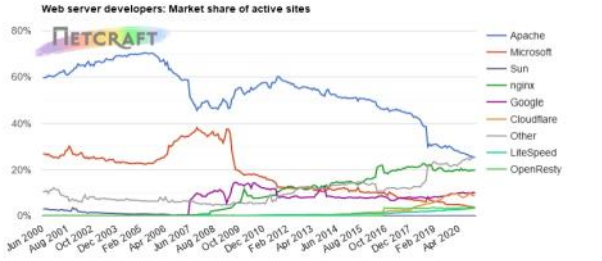
Main focus is performance (different technologies and architectures) and flexibility to integrate different programming languages to generate the requested page

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

Express is not one of the most popular! These are more mature, more secure, more fast!
JS is a single-threaded interpreted language! For a real site you need C code translation, full integration with server's OS to squeeze out all the cpu and network abilities of the server!

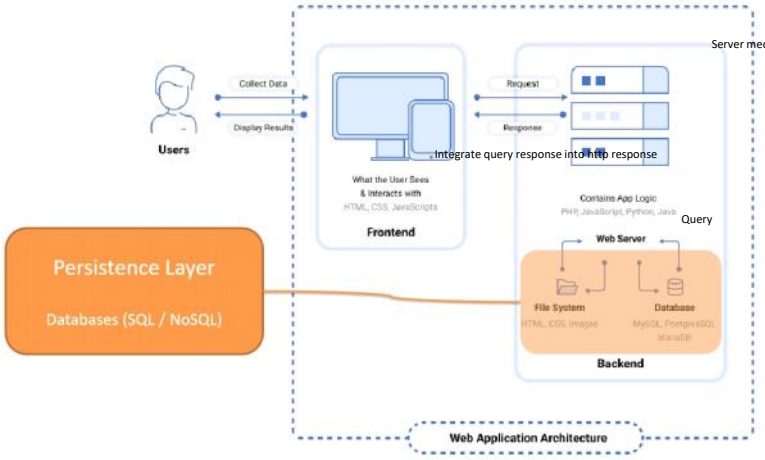
Web server with Node.js

Standard library

- 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

This other library is more functional, easy to learn, reasonably fast, integrated into node.

(changing very fast!)



Server mediates between browser (js code) and db

Integrate query response into http response

Query

Every piece needs different languages, technologies, architecture, requirements!

Web Architecture

ARCHITECTURAL PATTERNS

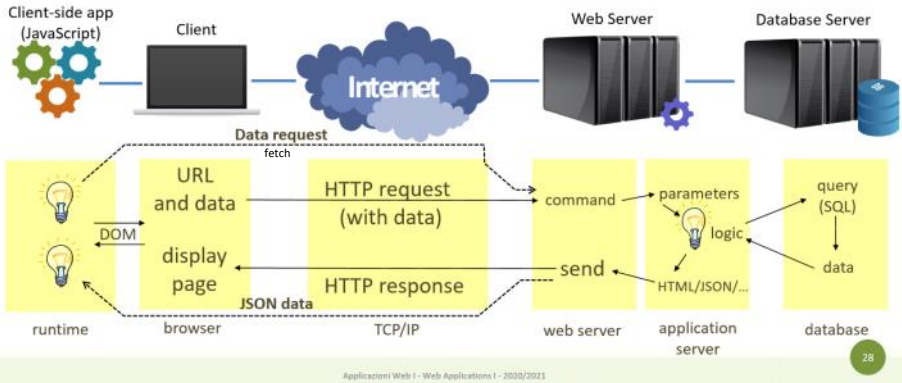
"Traditional" Architectural Pattern

5-10 years ago

- 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

Page by page (html) navigation
Each html page was generated on the server side using js

All The Layers At Work...



Requests ->
Pages or data
Response <-

Lightbulbs=logic

Modern Patterns

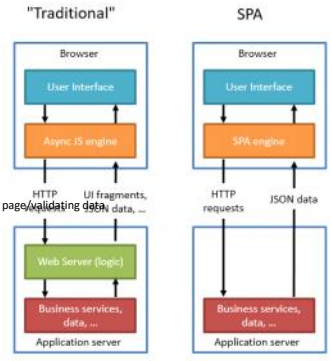
Other three patterns to architect a web application exist, roughly

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

Twitter.com response html is just an empty html with one js script which will ask for all the info it needs and will constuct the page directly on the client, so that the server will have lower load!

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



For moving from page to page/validating data

Changing from one page to another is just staying on the same page removing/adding new contents! Faster!

APIs over http to request/modify data on the server

Server is simplified!

SPA: Disadvantages

- SEO optimization 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!

Back and Forward button don't work as expected if not properly managed!

Client can see and modify the code that he is running, trying to break into your website! Protect server from being called by modified version of your code!

Modern Patterns

Other three patterns to architect a web application exist, roughly

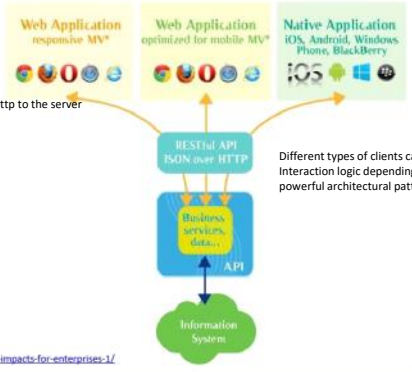
1. **Single-Page Application** (SPA)
   - the server sends the exact same web page for every unique URL
   - the page runs JavaScript to change the content and the aspect
   - by querying another (logical) server which provides "raw" information
2. **Isomorphic Application**
   - Combination of SPA with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

If I have the same language on server or client, both could run the same code to generate the page (traditional or single page approach), so you can decide when to use one or the other depending on the operation to do (save page, print page, index page by SEO, generate page for low computation client-> better generate on server; strong client, spa advantages-> better generate on client)

Good for mobile: website that looks like mobile application. Developed using browser technogies, but so well integrated with mobile OS that it looks like a native application. Same technology stack and knowhow, to create mobile application.

Supporting mobile development

Progressive



Mobile app runs browser that calls apis over http to the server

POLITO App

Different types of clients can share the same set of server side apis!
Interaction logic depending on application type, but only one server api that exposes data! Very powerful architectural patterns! 1 api used by many different applications!

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

Client-side, server-side, databases

Full control their variability!

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[13][14]
Yahoo	750,000,000	JavaScript	PHP	PostgreSQL, HBase, Cassandra, MongoDB,[15]
Amazon	500,000,000	JavaScript	Java, C++, Perl[16]	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]

!!! JS is the only supported language by the browser!

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/

Very interesting!!!

- 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