

# Introduction to Web Development

Part 1: Http – TCP/IP  
Client/Server – Rest  
Proxy, Gateway, Ng serve

[arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

Course Esilv 2023

This document:

<https://github.com/Arnaud-Nauwynck/presentations/web/intro-web-dev-part1-http.pdf>

# Outline

- http protocol,  
sample step by step on top of TCP-IP
- http Client Request – Server Response principles.  
Header extensibility
- Minimalist client/server application (java, then springboot)
- Typologies of servers  
static pages server, dynamic server-side pages, Rest API,  
Gateway, Proxy, angular « ng serve »

# Network Protocols: TCP-IP, DNS, Http

What happen on the network  
when you type in your Browser

« <https://www.google.com>»

« <https://github.com/Arnaud-Nauwynck?tab=repositories>»

« <http://localhost:8080/bar?query=xyz#baz>»

??

Guessed?  
« http://localhost:8080 » is NOT part of path  
« #bar » is skipped



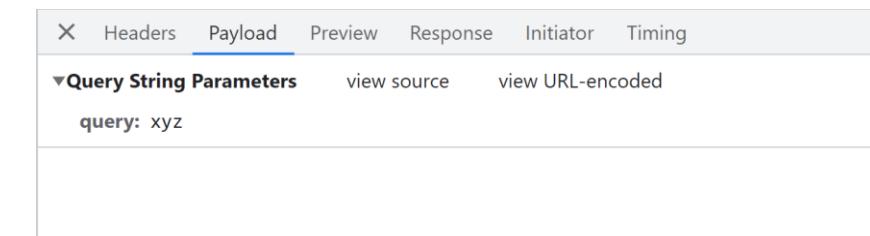
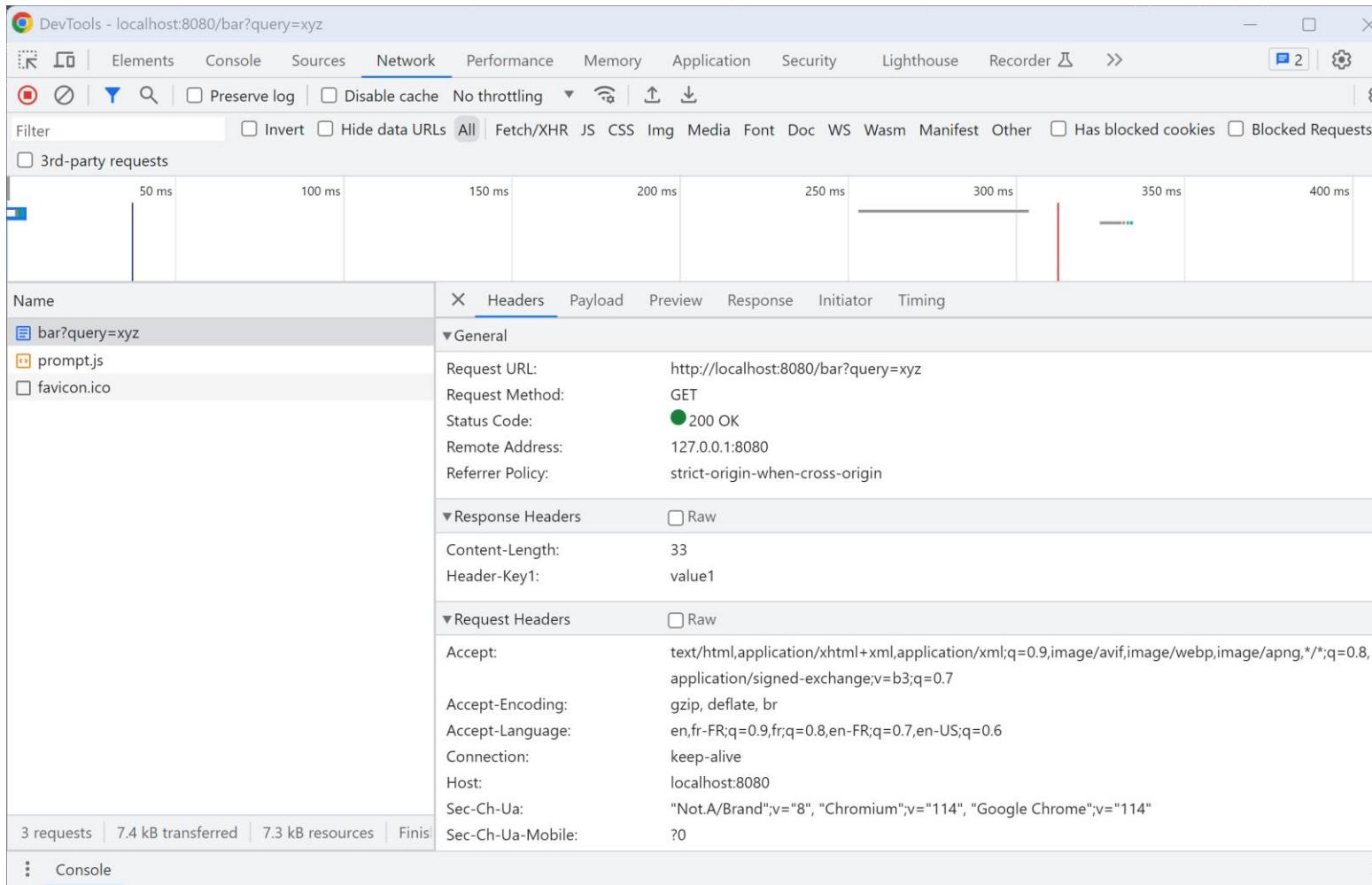
Interpreted because http is over TCP-IP

Interpreted by web browser (for page paragraph « #anchor »)

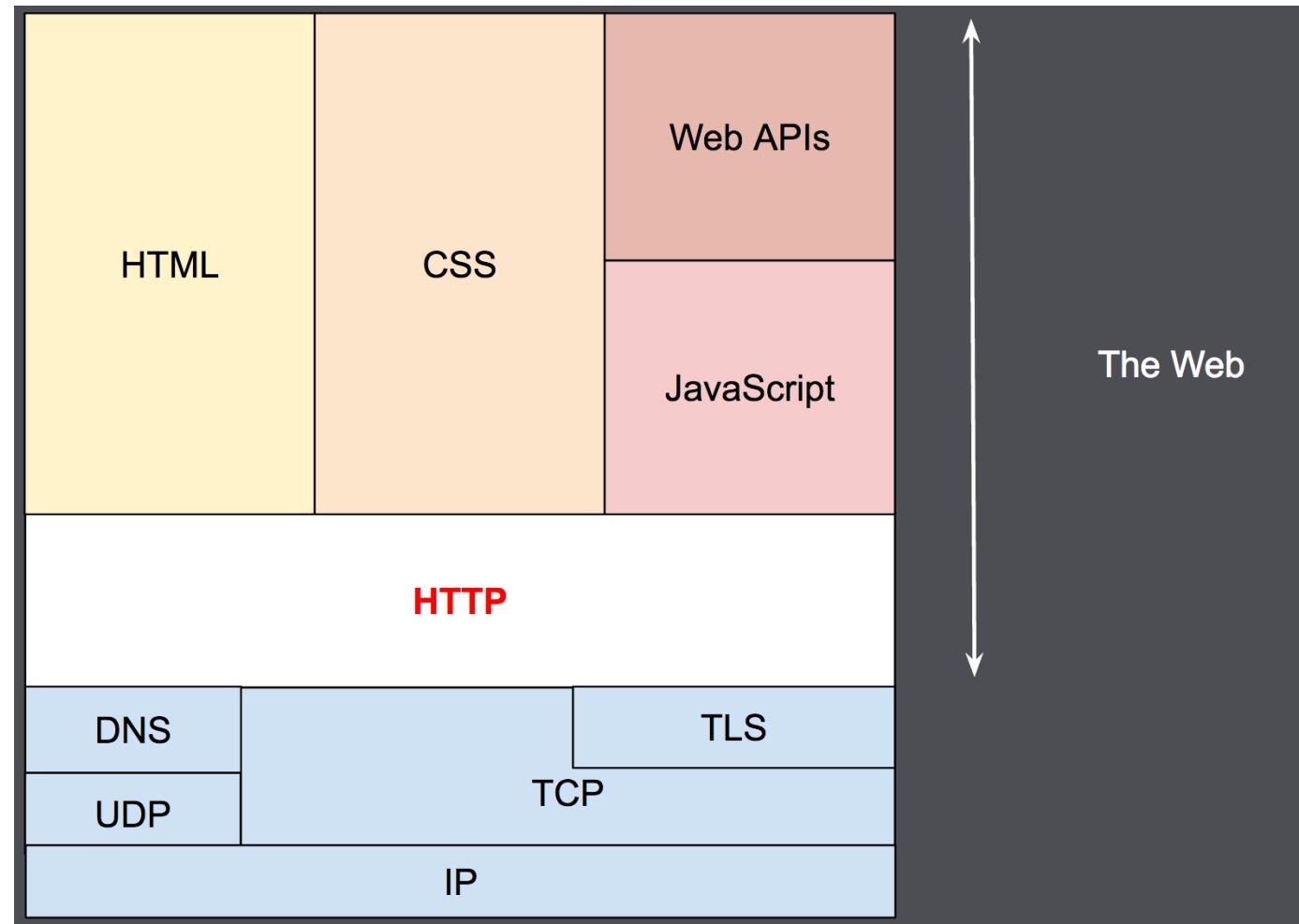


```
Cmder
$ curl -vv http://localhost:8080/bar?query=xyz#baz
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /bar?query=xyz HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.0.1
> Accept: */*
>
< HTTP/1.1 200 OK
< header-key1:value1
< content-length:33
<
<html><body> Hello </body></html>* Connection #0 to host localhost left intact
```

# F12 : Chrome Debugger Tool > Network

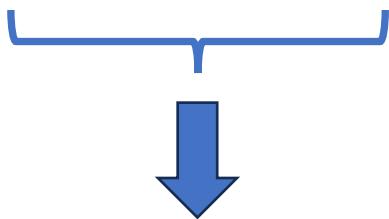


# <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>



# Step 1: URL is parsed, DNS resolved

`https://www.google.fr:443/bar?query=xyz#baz`



DNS: resolve hostname -> IP address

Equivalent to

**\$ nslookup www.google.fr**

```
Cmder
$ nslookup www.google.fr
Serveur : UnKnown
Address: 192.168.0.254

Réponse ne faisant pas autorité :
Nom : www.google.fr
Addresses: 2a00:1450:4007:818::2003
142.250.178.131

$
```

Step 2: resolve local IP address + gateway address  
+ create TCP-IP Socket

# Connection

from: localhost -> to: www.google.com



TCP-IP socket (=4 values in IP packets)

From IP: 192.168.0.10

fromPort:24352 (random)

To IP : 142.250.178.131

toPort: 443

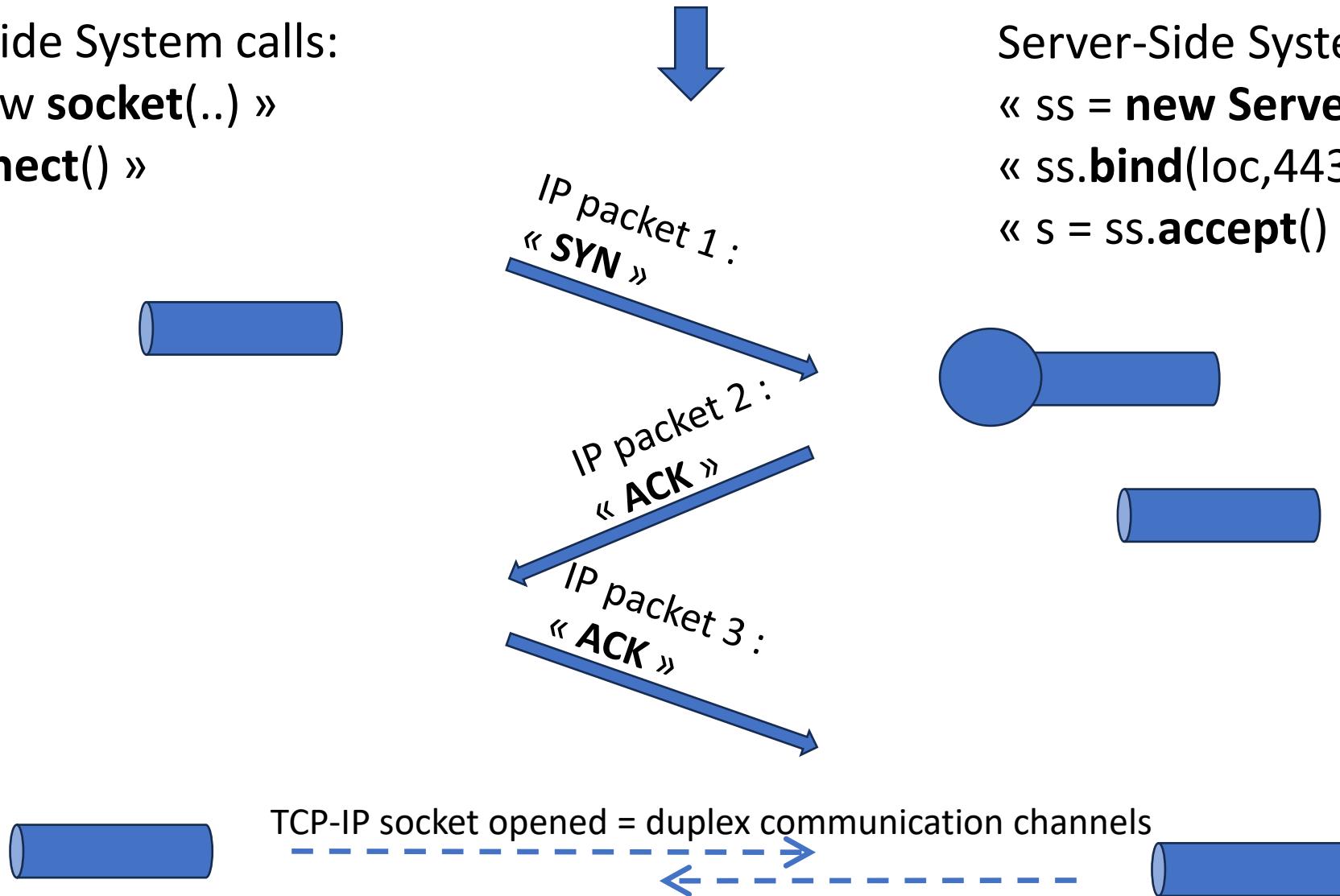
# Step 3: TCP-IP socket connection

Client-Side System calls:

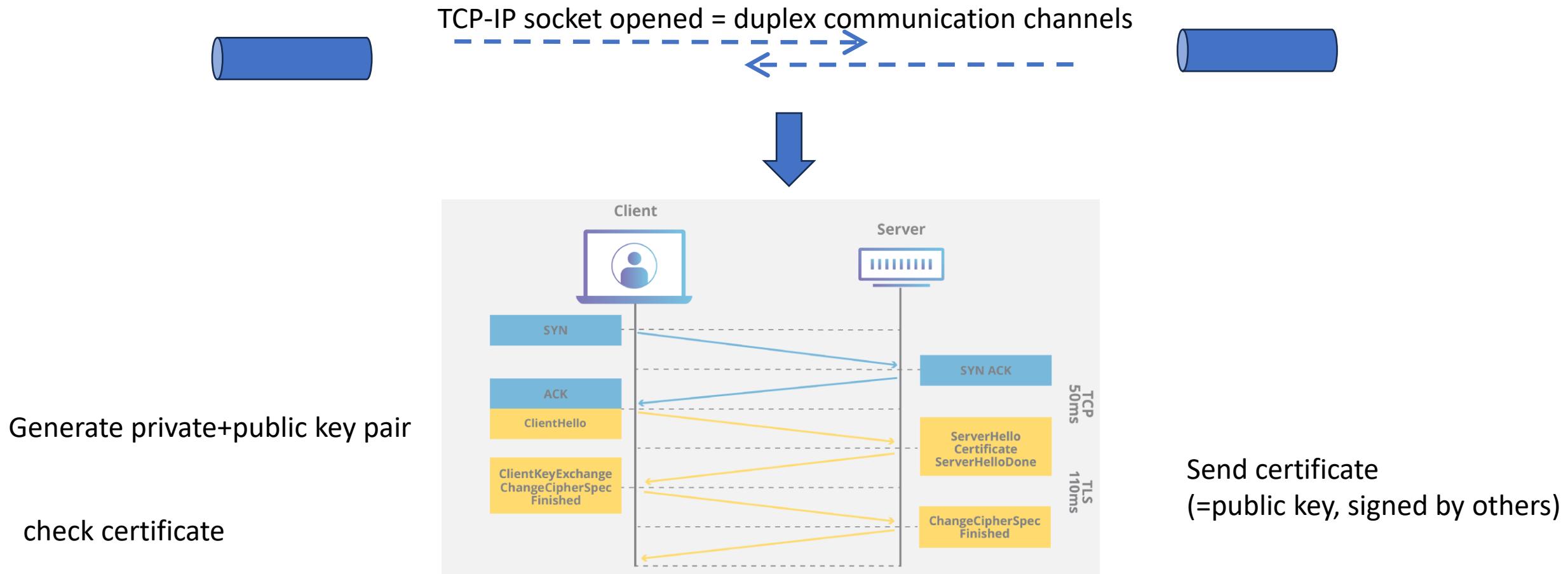
```
« s = new socket(..) »  
« s.connect() »
```

Server-Side System calls:

```
« ss = new ServerSocket() »  
« ss.bind(loc,443); »  
« s = ss.accept() »
```



# Step 4: https = http + TLS (SSL) => Handshake (generate private+pub key – exchange pub key..)



# Step 5 : client write http Request text

Client WRITE request text, then flush socket buffer



Line1 =

« GET /path/foo?bar HTTP/1.1 »

Line 2,3...N= http headers:

« key1 : value1 »  
« key2 : value2 »  
« keyN : valueN »

Empty Line: « »

Then : body... N bytes

« binary-content,  
maybe ascii html, json, image, ... »

# Step 5 : client write http Request text

Client WRITE request text, then flush socket buffer



Line1 =

« GET /path/foo?bar HTTP/1.1 »

Line 2,3...N= http headers:

« key1 : value1 »  
« key2 : value2 »  
« keyN : valueN »

Empty Line: « »

Then : body... N bytes

« binary-content,  
maybe ascii html, json, image, ... »

GET, PUT, POST, DELETE, HEAD, PATCH,...  
are http « **Method Verbs** »

**request Path** (and query parameters)  
are relative to server host

**http Request Headers**

some are standards headers: Host, Content-Type, Accept, ..

**http Request Body**

optional (generally none in GET and DELETE,  
one in PUT and POST)

Step 6: Client WAIT response,  
and do not write any more to socket

possible strategies:

- 1/ negotiate in http Header to keep socket open, and re-use
- 2/ half-close the socket on client-side (socket is now read-only)

# Step 7 : server write http Response



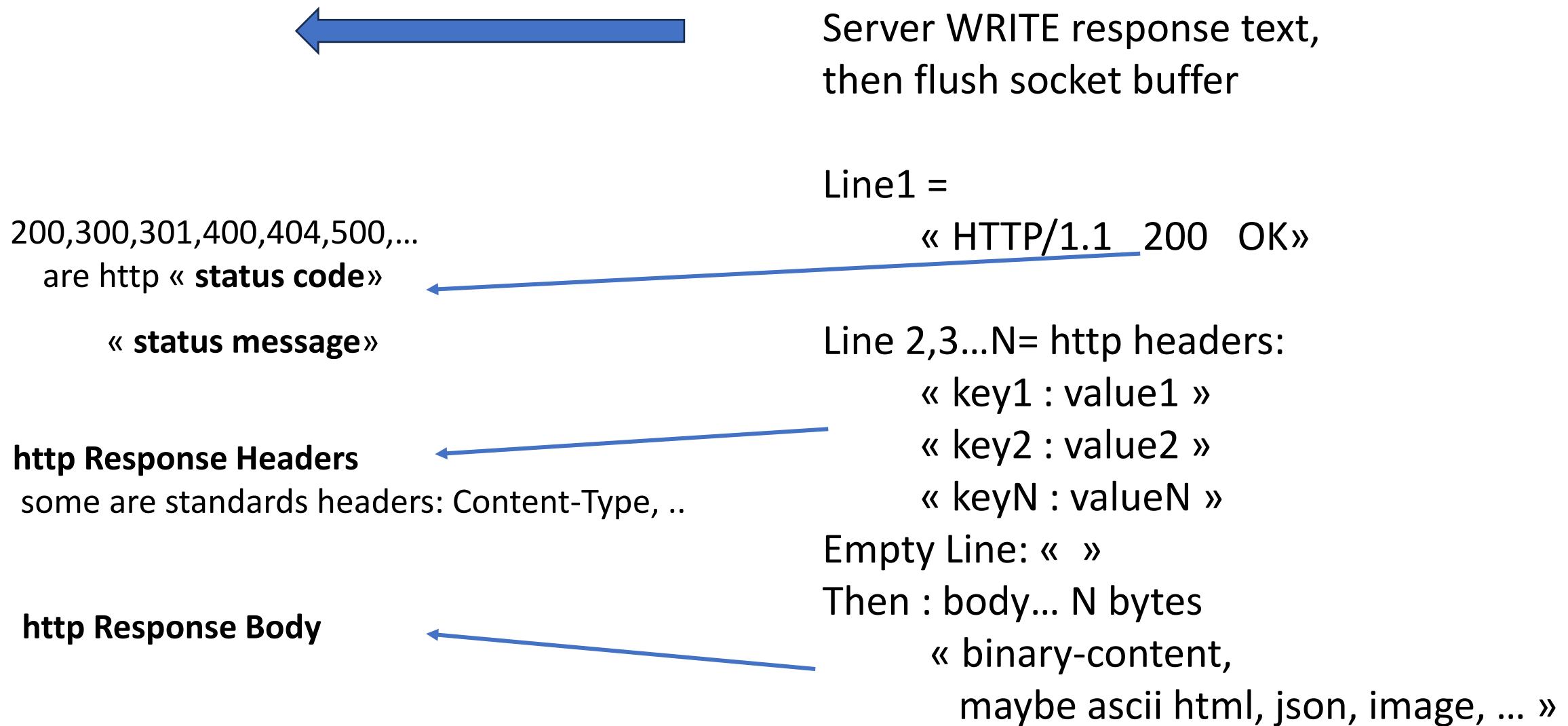
Server WRITE response text,  
then flush socket buffer

Line1 =  
« HTTP/1.1 200 OK»

Line 2,3...N= http headers:  
« key1 : value1 »  
« key2 : value2 »  
« keyN : valueN »

Empty Line: « »  
Then : body... N bytes  
« binary-content,  
maybe ascii html, json, image, ... »

# Step 7 : server write http Response



# Step 8: Client read response, close socket



# http(s) Protocol Characteristics

http 1.x protocol is human readable,  
( http 2 ... encapsulate multiple connections in frames )

http is Simple

http is Stateless (no session)

http is Extensible

https = http over TLS is secure

# http protocol: Simple text headers - body

http {method} {/path} HTTP/1.1

header1=value1

...

headerN=valueN

<body>



HTTP/1.1 {statusCode} {statusMessage}

header1=value1

...

headerN=valueN

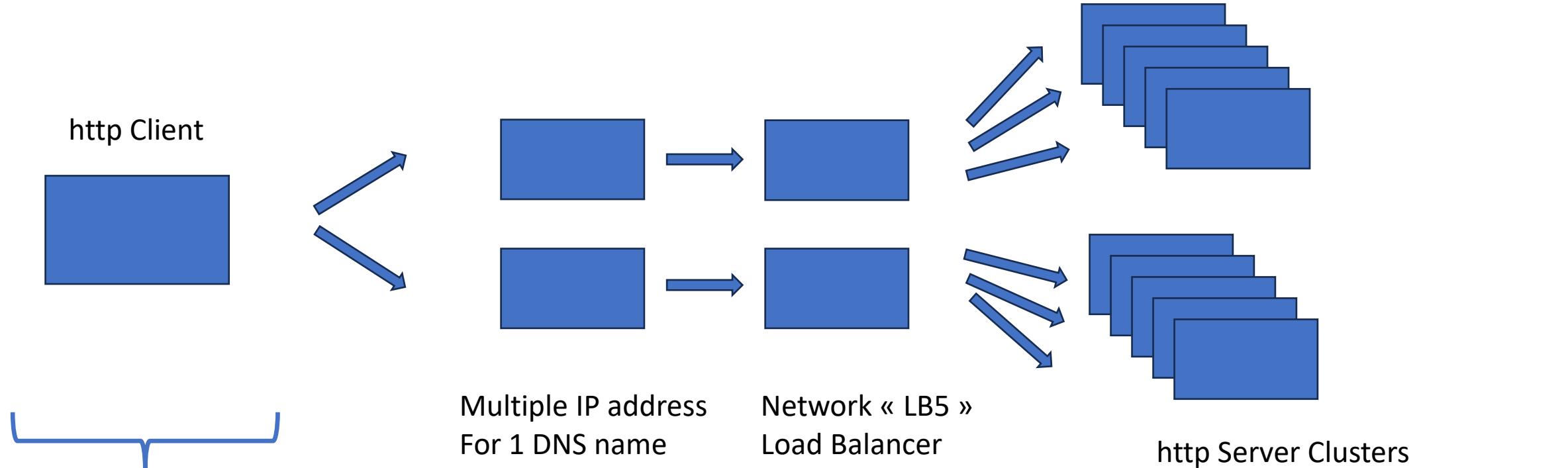
<body>

# HTTP is Stateless

N Calls => create N independent sockets !!!  
N http requests !!!

Requests can pass « session » information  
in /path/?sessionId={1234567}  
in httpHeader: header-sessionId={1234567}

# Stateless = Fundamentals for Load-Balancing ( resilience + horizontal scalability)



You are not alone on Internet  
... there can be Billions http clients !!!

Example: Route53 on AWS

Example: Ingress on Kubernetes

# Connection latency ... HTTP socket re-use

Establishing a https connection is « SLOW »

- Need 3 IP packets for TCP: SYN-ACK-ACK
- Then ... for TLS Handshake

Socket may be re-used for performance:

cf http header « keep-alive »

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive>

Cf also Http2: multiplex N simultaneous calls on 1 socket

# Http is Extensible !!

Headers can be used both in http Request and http Response

Not constrained by spec !! Fully negotiable between client and server

Unrecognized headers => OK, ignored, but NOT an error

Standard headers in http,

Example: caching, compression, cookie, content-type, language preference, ...

# Sample minimalist Java program : Client part

```
public class HttpClientMain {  
    public static void main(String[] args) throws IOException {  
        // Step 1: connect  
        Socket socket = new Socket();  
        SocketAddress socketAddress = new InetSocketAddress("localhost", 8080);  
        socket.connect(socketAddress);  
  
        // Step 2: write request  
        OutputStream socketOutput = socket.getOutputStream();  
        String requestText = "GET /foo HTTP/1.1\\n"  
            + "header1:value1\\n"  
            + "\\n" // empty line  
            ; // no request body here  
        socketOutput.write(requestText.getBytes());  
        socketOutput.flush();  
  
        // Step 3: read response  
        InputStream socketInput = socket.getInputStream();  
        BufferedReader socketLineReader = new BufferedReader(  
            new InputStreamReader(socketInput));  
        for(;;) {  
            String line = socketLineReader.readLine();  
            if (line == null) {  
                break;  
            }  
            System.out.println(line);  
        }  
        socket.close();  
    }  
}
```

# Sample minimalist Java program: Server part

```
public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket();
    serverSocket.bind(new InetSocketAddress("localhost", 8080));
    for(;;) {
        Socket socket = serverSocket.accept();
        System.out.println("accepted socket connection from "
            + socket.getInetAddress() + ":" + socket.getPort());
        InputStream socketInput = socket.getInputStream();
        BufferedReader socketReader = new BufferedReader(
            new InputStreamReader(socketInput));
        String reqLine = socketReader.readLine();
        for(;;) {
            String headerLine = socketReader.readLine();
            if (headerLine == null) {
                break;
            }
            if (headerLine.isEmpty()) {
                // no more http header
                break;
            }
        }
        // read request body.. Not needed here
        // write response
        OutputStream socketOutput = socket.getOutputStream();
        String responseBody = "<html><body> Hello </body></html>";
        String responseText = "HTTP/1.1 200 OK\n"
            + "header-key1:value1\n"
            + "content-length:" + responseBody.length() +"\n"
            + "\n" // empty
            + responseBody;
        socketOutput.write(responseText.getBytes());
        socketOutput.flush();
        socketOutput.close();
        socket.close();
    }
}
```

# Run it ...

Launch Server

Open Chrome Web Browser

Type url: <http://localhost:8080/>



Hello



# F12 (Chrome Debugger Tools > Network )

The screenshot shows the Chrome DevTools Network tab for the URL `localhost:8080/index.html`. The timeline at the top indicates a total duration of 500 ms for the request. The request for `index.html` is highlighted in blue, showing a duration of approximately 300 ms.

**Name:** index.html

**General**

- Request URL: `http://localhost:8080/index.html`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 127.0.0.1:8080
- Referrer Policy: strict-origin-when-cross-origin

**Response Headers**

- Content-Length: 33
- Header-Key1: value1

**Request Headers**

- Accept: `text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7`
- Accept-Encoding: gzip, deflate, br
- Accept-Language: en,fr-FR;q=0.9,fr;q=0.8,en-FR;q=0.7,en-US;q=0.6
- Cache-Control: max-age=0
- Connection: keep-alive
- Host: localhost:8080
- Sec-Ch-Ua: "Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"
- Sec-Ch-Ua-Mobile: ?0
- Sec-Ch-Ua-Platform: "Windows"
- Sec-Fetch-Dest: document

At the bottom, it shows 3 requests, 7.4 kB transferred, 7.3 kB resources, and a finished status.

# Debug Client & Server - Logs

The screenshot shows the IntelliJ IDEA debugger interface with two tabs: "HttpServerMain" and "HttpClientMain". The "HttpClientMain" tab is active, displaying the following log output:

```
C:\apps\jdk\jdk-17.0.1.12\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=1  
Connected to the target VM, address: '127.0.0.1:57712', transport: 'socket'  
HTTP/1.1 200 OK  
header-key1=value1  
content-length=33  
  
<html><body> Hello </body></html>  
Disconnected from the target VM, address: '127.0.0.1:57712', transport: 'socket'  
  
Process finished with exit code 0
```

The screenshot shows the IntelliJ IDEA debugger interface with two tabs: "HttpServerMain" and "HttpClientMain". The "HttpServerMain" tab is active, displaying the following log output:

```
C:\apps\jdk\jdk-17.0.1.12\bin\java.exe -agentlib:jdwp=transpor  
Connected to the target VM, address: '127.0.0.1:57703', transpor  
accepted socket connection from /127.0.0.1:57714  
accepted socket connection from /127.0.0.1:57914  
accepted socket connection from /127.0.0.1:57915  
accepted socket connection from /127.0.0.1:57916
```

# Test using Curl (or Postman)

```
$
$ curl http://localhost:8080
<html><body> Hello </body></html>$
$
$ curl -vv http://localhost:8080
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.0.1
> Accept: */*
>
< HTTP/1.1 200 OK
< header-key1:value1
< content-length:33
<
<html><body> Hello </body></html>* Connection #0 to host localhost left intact
$
```

ANNEXE ...  
equivalent Server code using frameworks

# Minimalist Java Server ... using SpringBoot = 1 line of code !!

Actually more « <xml> </xml> » in maven pom.xml  
than in java

And more java « @annotation »  
than java « {code} » !!!

<https://start.spring.io/>

# <https://start.spring.io/>

## add dependency « WEB »

start.spring.io

Meet the Spring team this August at SpringOne.

### spring initializr

**Project**

Gradle - Groovy  
 Gradle - Kotlin  Maven

**Language**

Java  Kotlin  Groovy

**Spring Boot**

3.1.2 (SNAPSHOT)  3.1.1  3.0.9 (SNAPSHOT)  3.0.8  
 2.7.14 (SNAPSHOT)  2.7.13

**Project Metadata**

Group com.example  
Artifact demo  
Name demo  
Description Demo project for Spring Boot

**Dependencies**

**Spring Web** **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**ADD DEPENDENCIES... CTRL + B**

GENERATE CTRL + ↵ EXPLORE CTRL + SPACE SHARE...

Q Twitter

# Click « Generate », Unzip, Import in IDE, Run

The screenshot shows the IntelliJ IDEA interface with a Java Spring Boot project named "demo". The code editor displays the main class `DemoApplication.java`:

```
package com.example.demo;  
import ...  
@SpringBootApplication  
public class DemoApplication {  
    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }  
}
```

The Project tool window on the left shows the directory structure:

- demo
- .idea
- .mvn
- src
  - main
    - java
      - com.example.demo
    - resources
      - static
        - hello.html
      - templates
      - application.properties
  - test
  - target

The Debug tool window at the bottom shows the application's startup logs:

```
2023-07-03T09:50:48.548+02:00 INFO 9616 --- [           main] com.example.demo.DemoApplication      : Starting DemoApplication using Java 20.0.1 with PID 9616 (C:\arn\devPers...  
2023-07-03T09:50:48.555+02:00 INFO 9616 --- [           main] com.example.demo.DemoApplication      : No active profile set, falling back to 1 default profile: "default"  
2023-07-03T09:50:50.146+02:00 INFO 9616 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)  
2023-07-03T09:50:50.162+02:00 INFO 9616 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]  
2023-07-03T09:50:50.162+02:00 INFO 9616 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.10]  
2023-07-03T09:50:50.364+02:00 INFO 9616 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext  
2023-07-03T09:50:50.370+02:00 INFO 9616 --- [           main] w.s.c.ServletWebServerApplicationContext  : Root WebApplicationContext: initialization completed in 1710 ms  
2023-07-03T09:50:51.162+02:00 INFO 9616 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''  
2023-07-03T09:50:51.181+02:00 INFO 9616 --- [           main] com.example.demo.DemoApplication        : Started DemoApplication in 3.295 seconds (process running for 4.622)  
2023-07-03T09:51:10.268+02:00 INFO 9616 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'  
2023-07-03T09:51:10.274+02:00 INFO 9616 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'  
2023-07-03T09:51:10.280+02:00 INFO 9616 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
```

The status bar at the bottom indicates "Process started" and the current time as 24:1.

# Coffee Break

## http Take Away

http(s) is a Simple protocol on top of TCP-IP

Extensible by Headers

Simple to write a Client / Server program

Widely used ... The core of Web

# More on http ...

## Theory:

Some Standard status code / headers imposed by W3C spec  
Mostly FREE to choose URL, Header and Content

## Practices:

How it is used in real life

Sample « type of servers »: Static CDN Pages, Dynamic Server Pages,  
http Api Server (Rest json servers), Frontend/Backend N-Tiers architecture

Rest, web naming conventions & Best practices

Gateway, Virtual Hosts, Proxy, Security Proxy ...

# Status Code Family: 200, 300, 400, 500

200, 201, 202 ... = OK

300, 301, 302 ... = maybe ok, but temporary unavailable, redirected..

400, 401, 402 ... = ERROR on client-side

500, 501, 502 ... = ERROR on server-side

# Common http Status Codes

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

303 = redirecting ... example to login page (for authentication)

304 = not modified

401 = unauthorized (unauthenticated)

403 = Forbidden

404 = Not Found (bad url or resource not exist)

500 = internal server error

502 = Bad Gateway

503 = Service Unavailable

http Method Verbs :  
GET, PUT, POST, DELETE, HEAD, PATCH, ..

GET = readonly to get a resource content  
(idempotent : can retry)

PUT = update an existing resource  
(idem potent: will overwrite/update resource)

POST = create a new resource  
(NOT idem potent: create a new one each time)

## http Method Verbs (bis): GET, PUT, POST, DELETE, HEAD, PATCH, ..

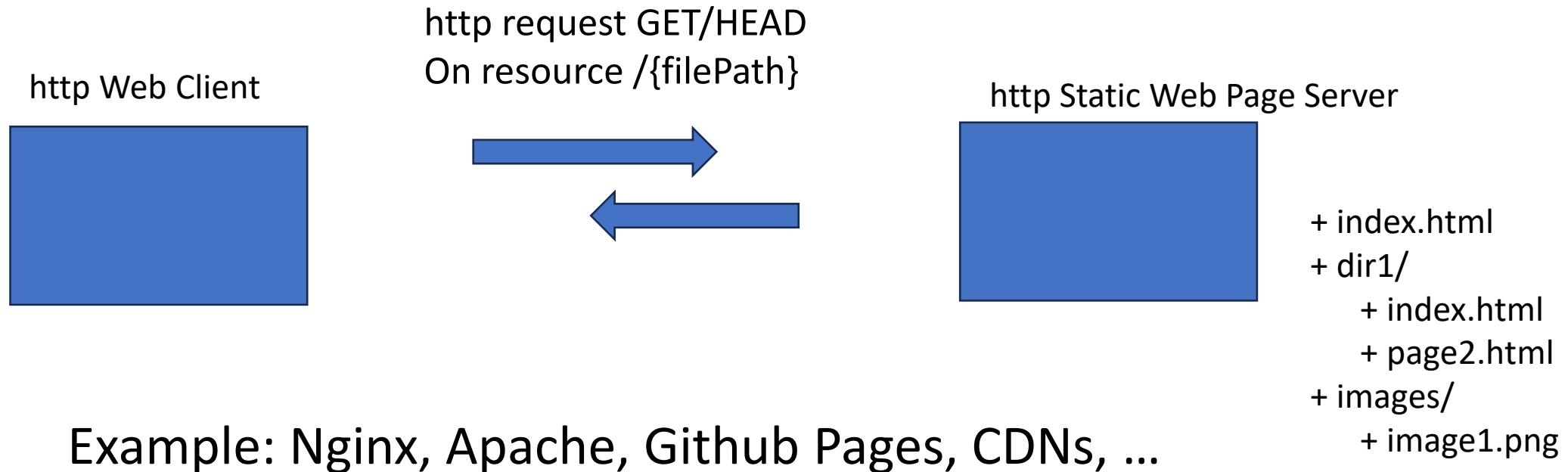
DELETE = delete a resource by its id/path  
(idempotent : can retry)

HEAD = only get http headers corresponding to a resource  
(idem potent) ... fast checking if exist/modified since

PATCH = partial update of a resource  
(maybe NOT idem potent on field insert,  
or idem potent on field update)

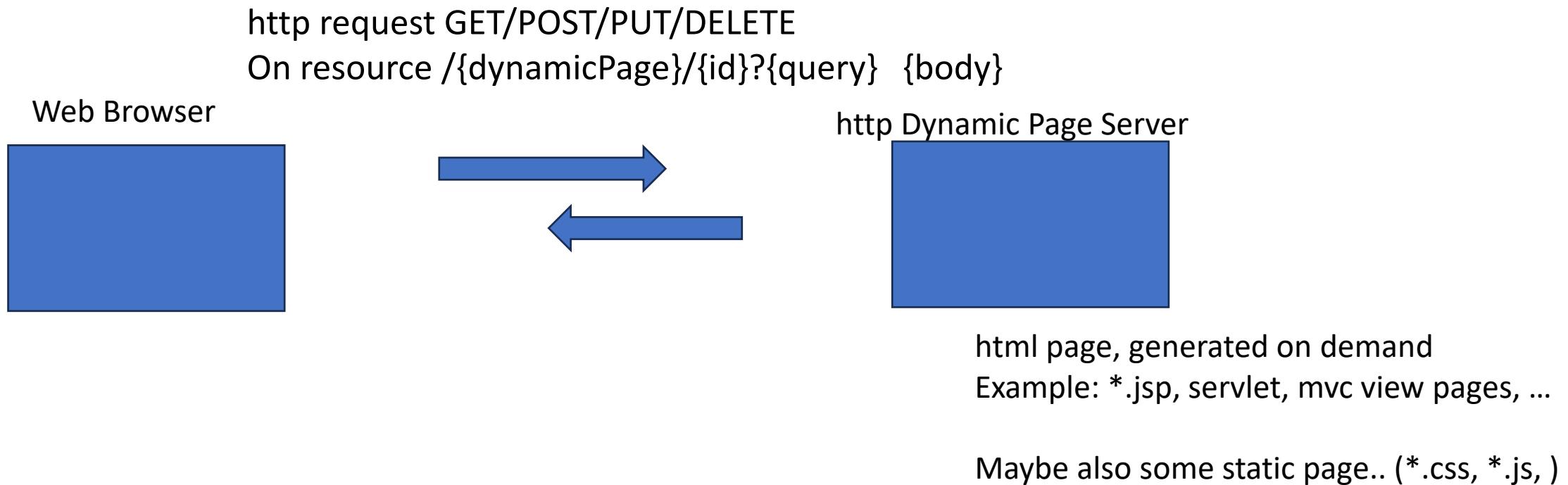
Simple case: http Rest Server responding  
ONLY \*.html/\* .js/\* .css/\* .png/...  
static page files

## Terminology: **Static Web Page Server**



# Dynamic Page Server ( server-side generated web page)

## Terminology: **Dynamic Page Server**



# Museum of Dynamic Page Server Technologies

/htdocs / cgi-bin / \*.sh

Index of /

Name	Last modified	Size	Description
cgi-bin/	2017-10-09 03:04	-	



**BASH**  
THE BOURNE AGAIN SHELL



Personal  
Home  
Page

(not a decent language)



\*.SP

**node** js  
express



# Java Servlet & Jsp

(html in java / java in pseudo-html )



Servlets

= plain old java code printing « html » text

```
void toHtml(ServletOutputStream out) {  
    if (cond) {  
        out.print("<div><span \"attr\"=\"v1\"> .. ");  
    } else {  
        out.print("<div><span \"attr\"=\"v2\"> .. ");  
    }  
  
out.print( new Date() );  
}
```



= pseudo html/xml/markup code containing java

```
<% if (cond) { %>  
    <div><span "attr"="v1"> .. </span></div>  
<% } else { %>  
    <div><span "attr"="v1"> .. </span></div>  
<% } %>
```

```
<%= new Date() %>
```

Choose your poison  
Red pill / blue pill



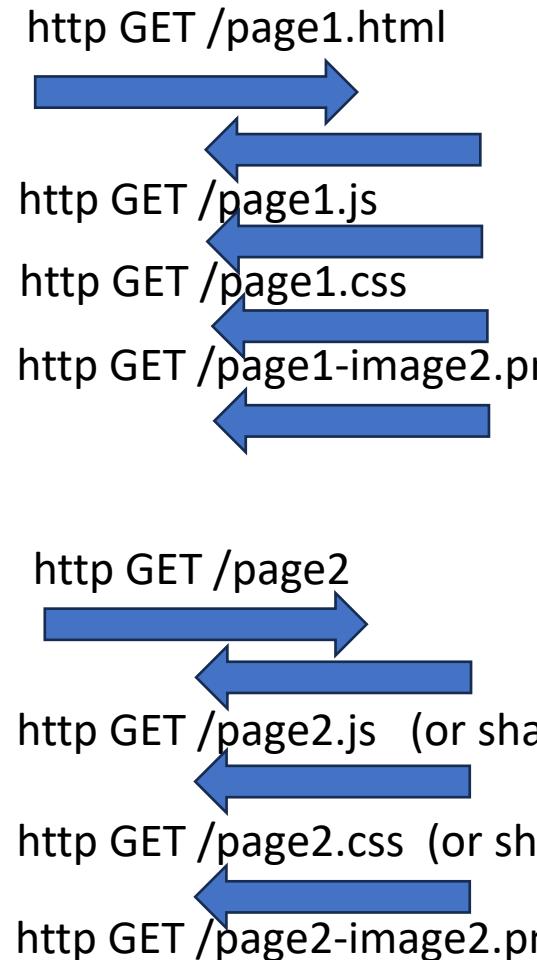
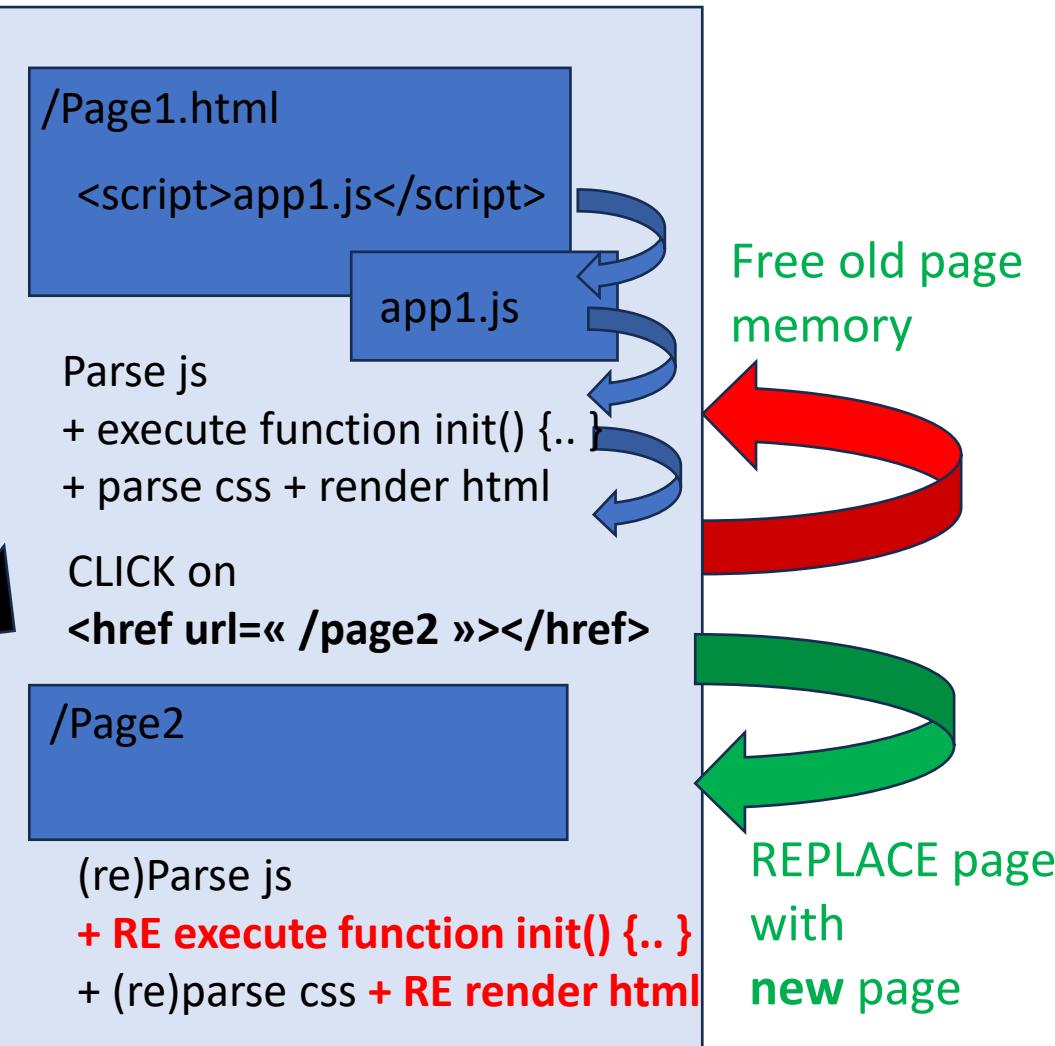
Not only a Developper problem...  
Html+Js+CSS re-Rendering slowness + Network + ..  
for reloading pages links

SLOW ... SLOW ....

NETWORK Bandwidth CONSUMING  
+ Network Latency  
+ CPU CONSUMING

# Click <href> ... RE-Loading new html page

Web Browser



Solution to  
**Multiple Pages** Application Problems ?

caching + optims + workarounds ... ?

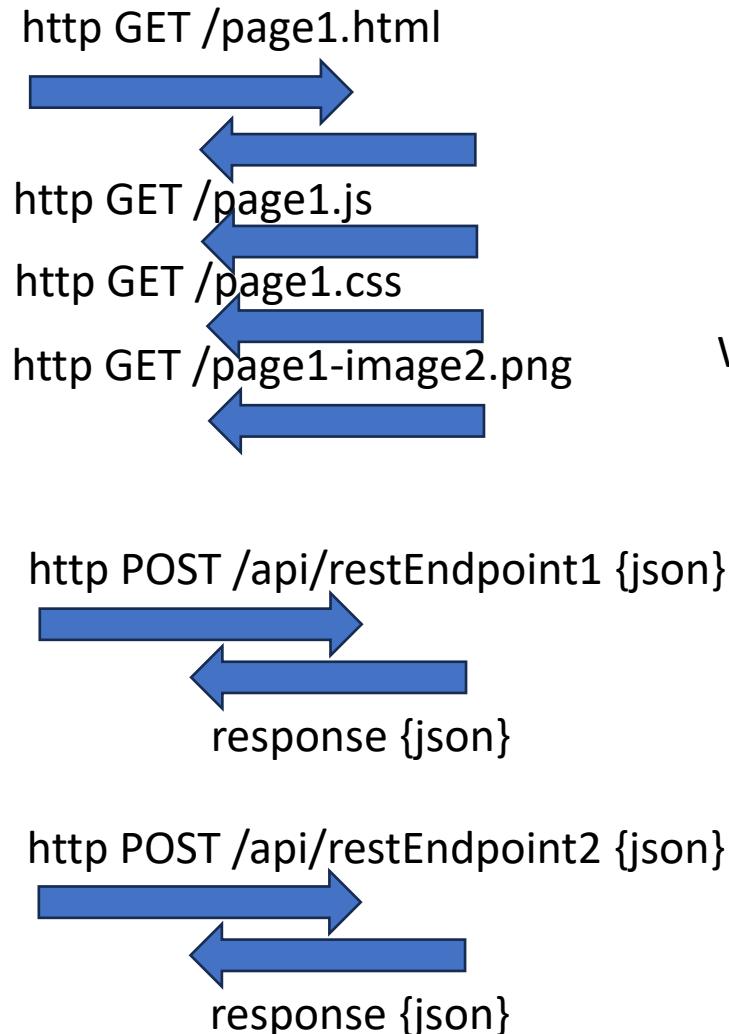
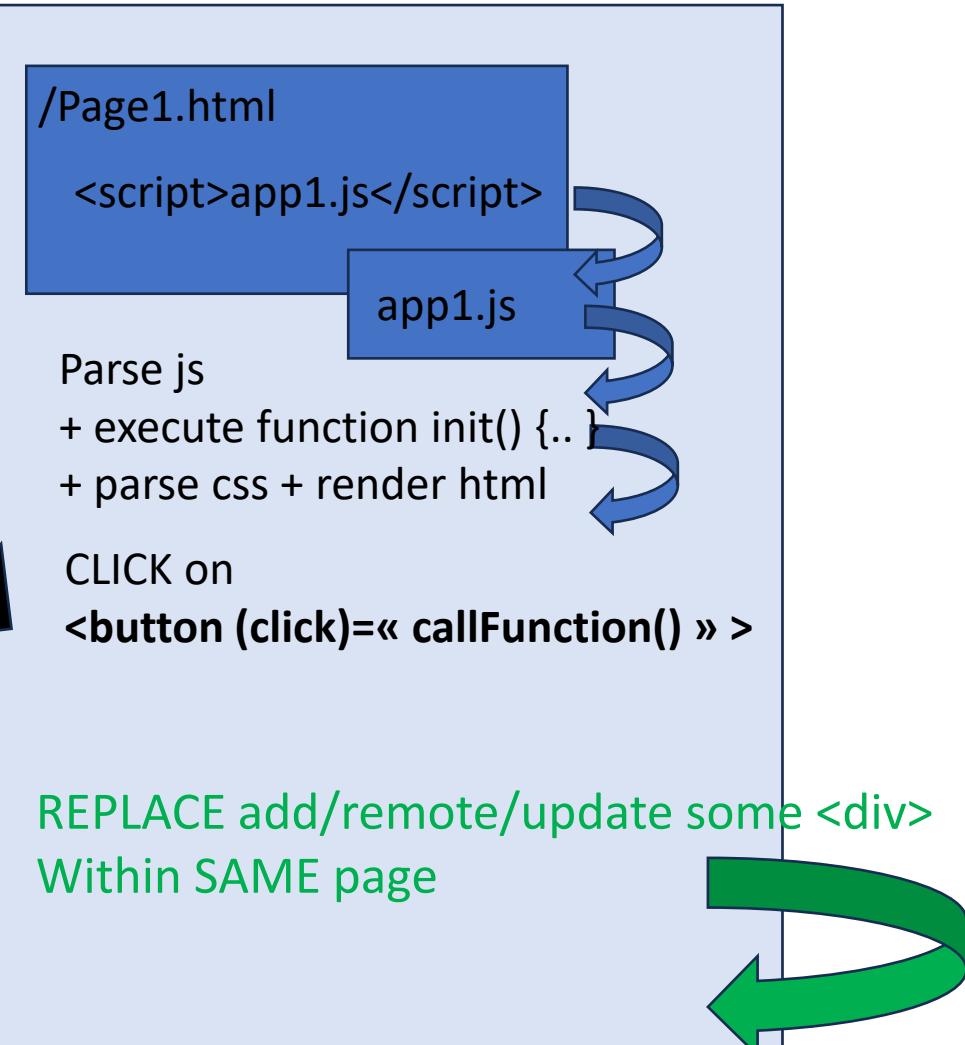
=>

SPA = **Single** Page Application !!

# SPA = Single Page Application

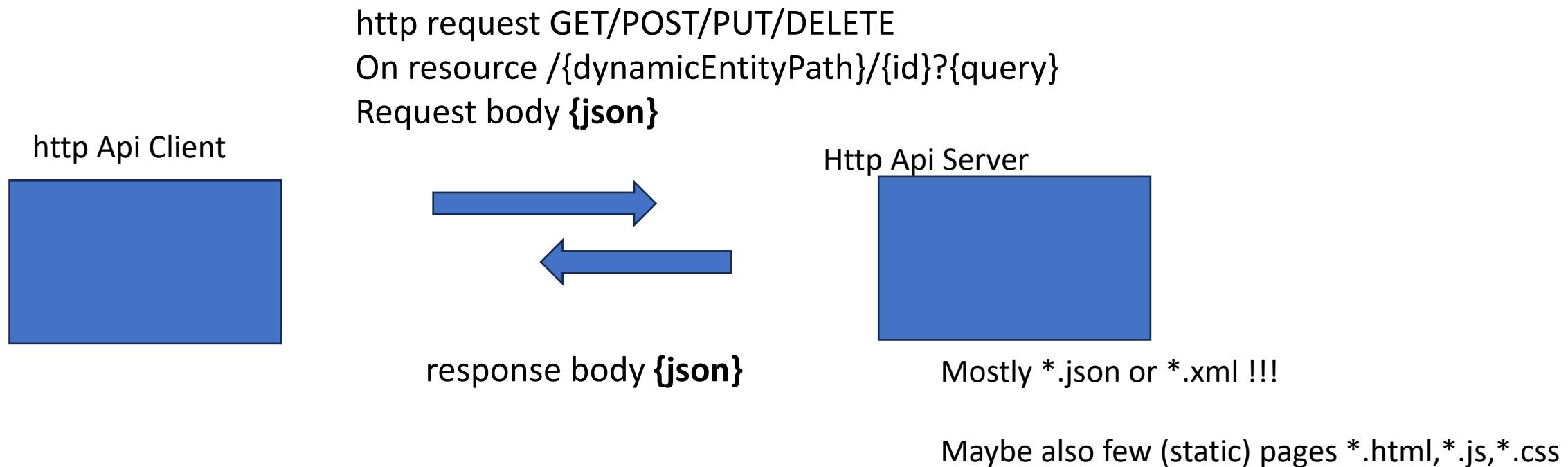
... still dynamic, still Multiple Rest Json requests

Web Browser



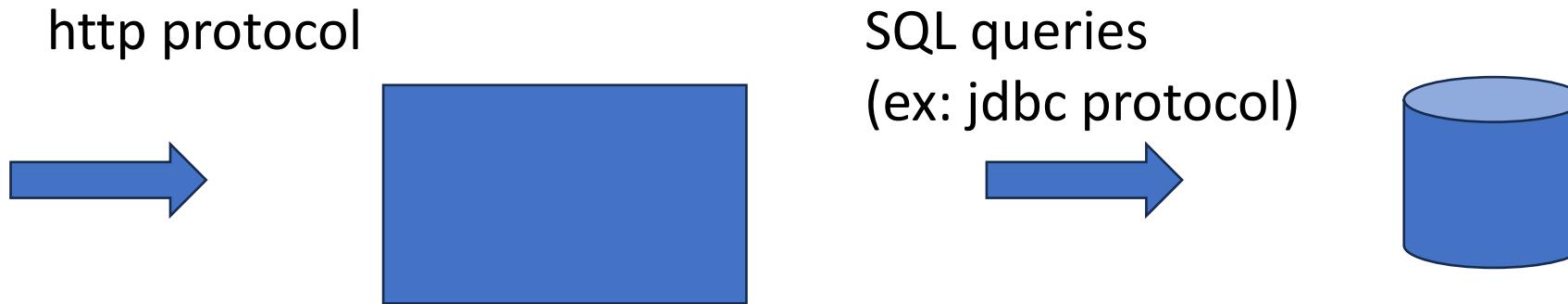
# http API Server (cf Rest Api)

## Terminology: **Http Api Server**



# http exposing a « CRUD » api

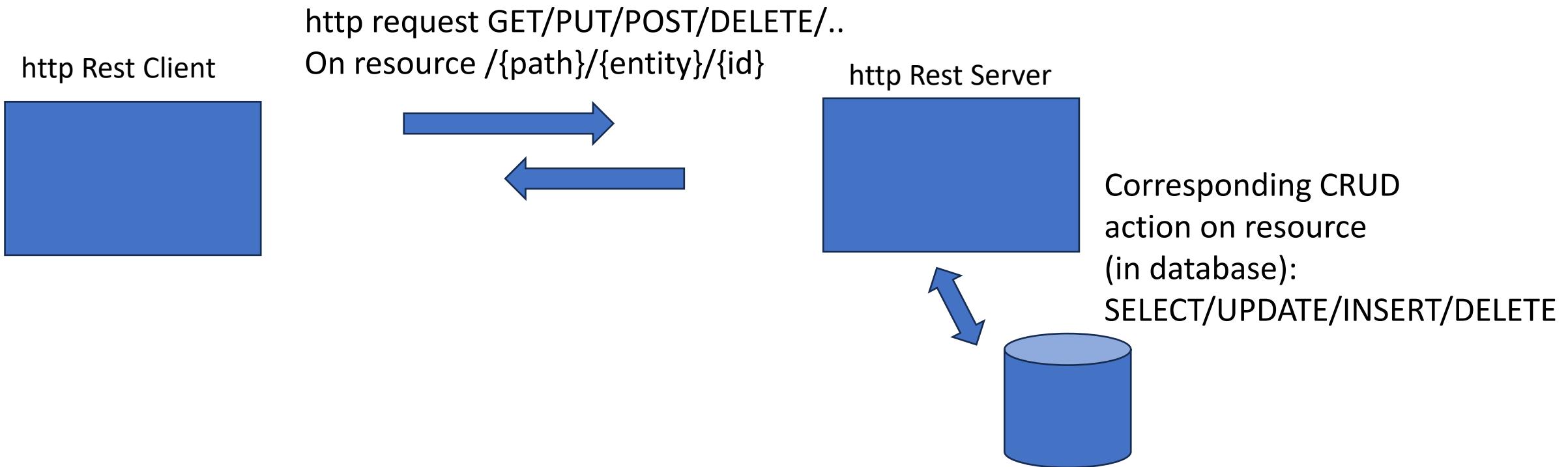
## CRUD : Create – Read - Update - Delete



Create	http <b>POST</b> /{entityType} ... body	<b>INSERT</b> into table T(..) values(..)
Read	http <b>GET</b> /{entityType}/{id}	<b>SELECT</b> * from T where id=?
	http <b>GET</b> /{entityType}?field=value	<b>SELECT</b> * from T where ... limit 10
Update	http <b>PUT</b> /{entityType}/{id} .. body	<b>UPDATE</b> T set ...=? where id=?
Delete	http <b>DELETE</b> /{entityType}/{id}	<b>DELETE</b> T where id=?

using Standard naming convention on Verb + Url  
= follow « Rest » style

Rest = **R**epresentational State Transfer



[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

en.wikipedia.org/wiki/Representational\_state\_transfer

WIKIPEDIA The Free Encyclopedia

Search Wikipedia

Search

Create account Log in ...

# Representational state transfer

29 languages

Contents [hide]

(Top)

Principle

History

Architectural properties

Architectural constraints

Client–server architecture

Statelessness

Cacheability

Layered system

Code on demand (optional)

Uniform interface

Classification models

Applied to web services

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

"REST" redirects here. For other uses, see [Rest](#).

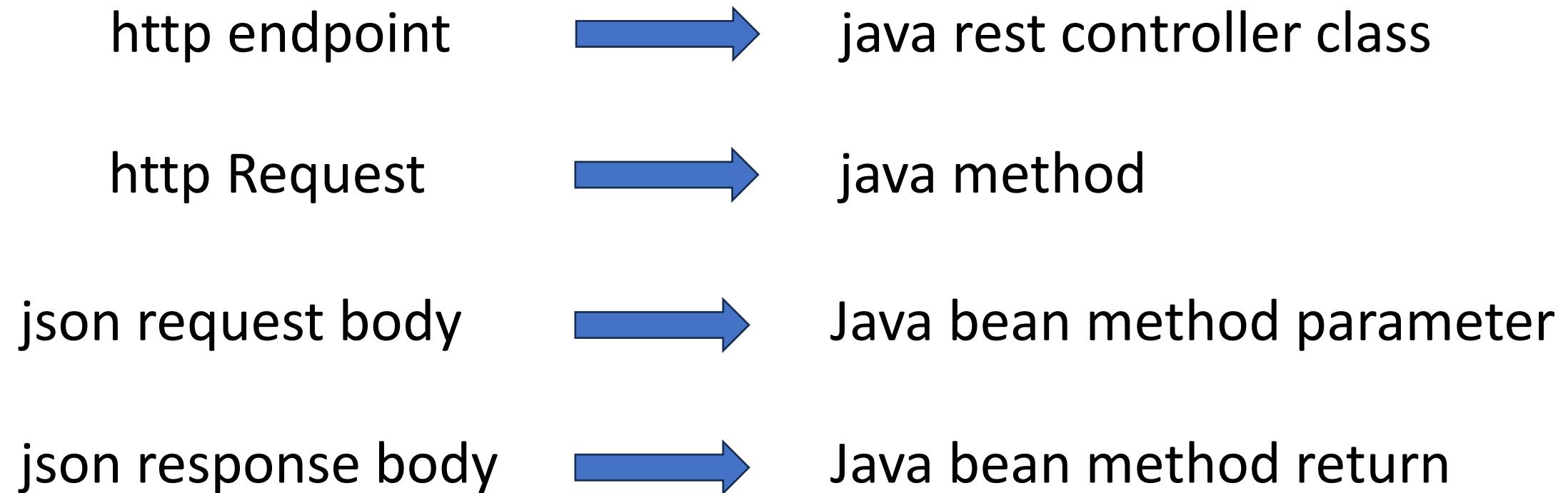
This article **may be too technical for most readers to understand**. Please [help improve it](#) to make it understandable to non-experts, without removing the technical details. (October 2020) ([Learn how and when to remove this template message](#))

**Representational state transfer (REST)** is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, should behave. The REST architectural style emphasises the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching of components to reduce user-perceived [latency](#), enforce [security](#), and encapsulate [legacy systems](#).<sup>[1]</sup>

REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating [stateless](#), reliable [web APIs](#). A web API that obeys the [REST constraints](#) is informally described as *RESTful*. RESTful web APIs are typically loosely based on [HTTP methods](#) to access [resources](#) via [URL-encoded](#) parameters and the use of [JSON](#) or [XML](#) to transmit data.

# Rest-Json Api server in Springboot

## @Mapping ..



# Example @RequestMapping

http PUT /api/endpoint1/meth2  
header1:value1

```
{  
    « reqField »: « value »  
}
```



http 200 OK  
header2:value2

```
{  
    « respField »: « value »  
}
```

```
class RequestDTO {  
    public String reqField;  
}
```

Json -> java



```
@RestController  
@RequestMapping(« /api/endpoint1 »)  
public class MyRestController {
```

```
@PutMapping(« meth2 »)  
public ResponseDTO handle(  
    @RequestBody RequestDTO req) {  
    return new ...  
}
```

Java -> json

```
class ResponseDTO {  
    public String respField;  
}
```

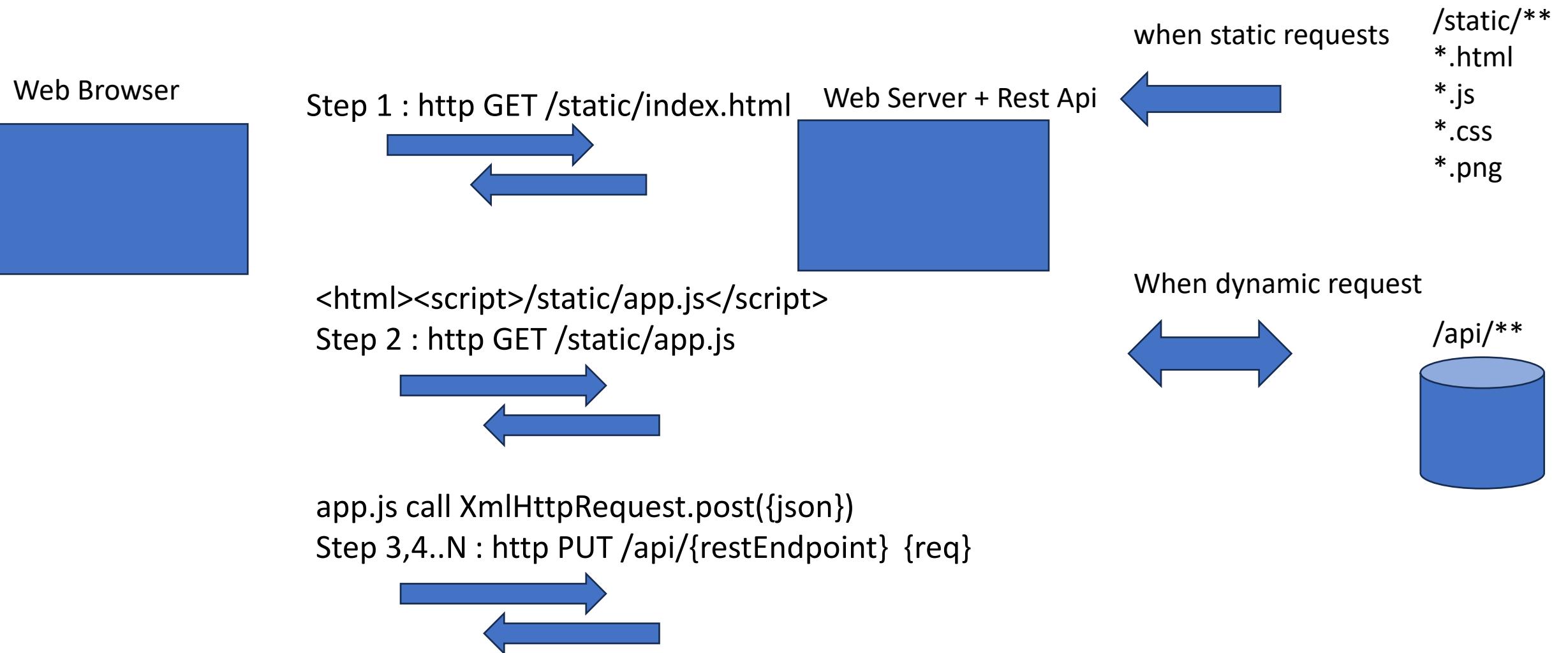
# Example using NodeJs - Express

```
const express = require('express');
const app = express();

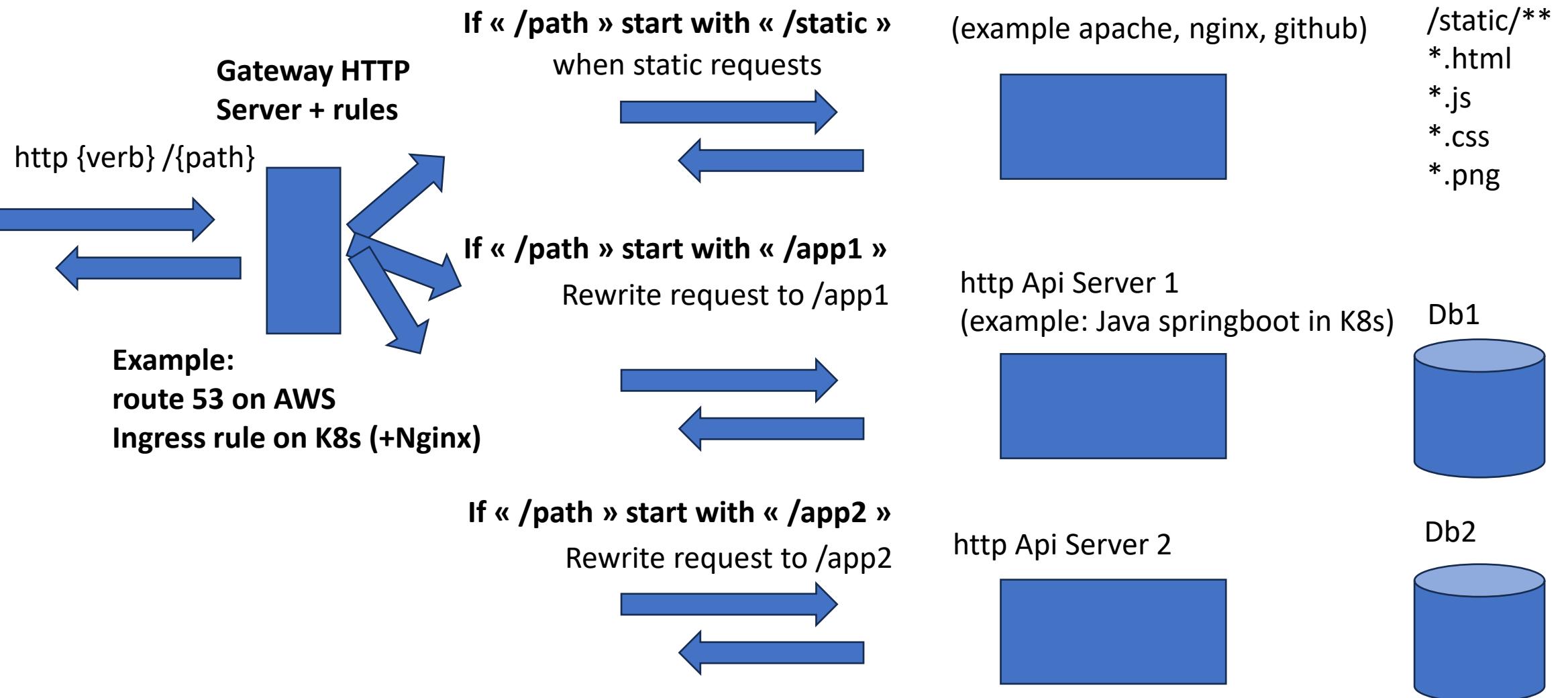
app.get('/', function (req, res) {
  res.send({ some: 'Hello Json Express' })
});

app.listen(3000);
```

# Mixing Static and Dynamic Content



# Gateway ... Ingress Rule



# in Angular Development Environment ..

## « Ng serve »

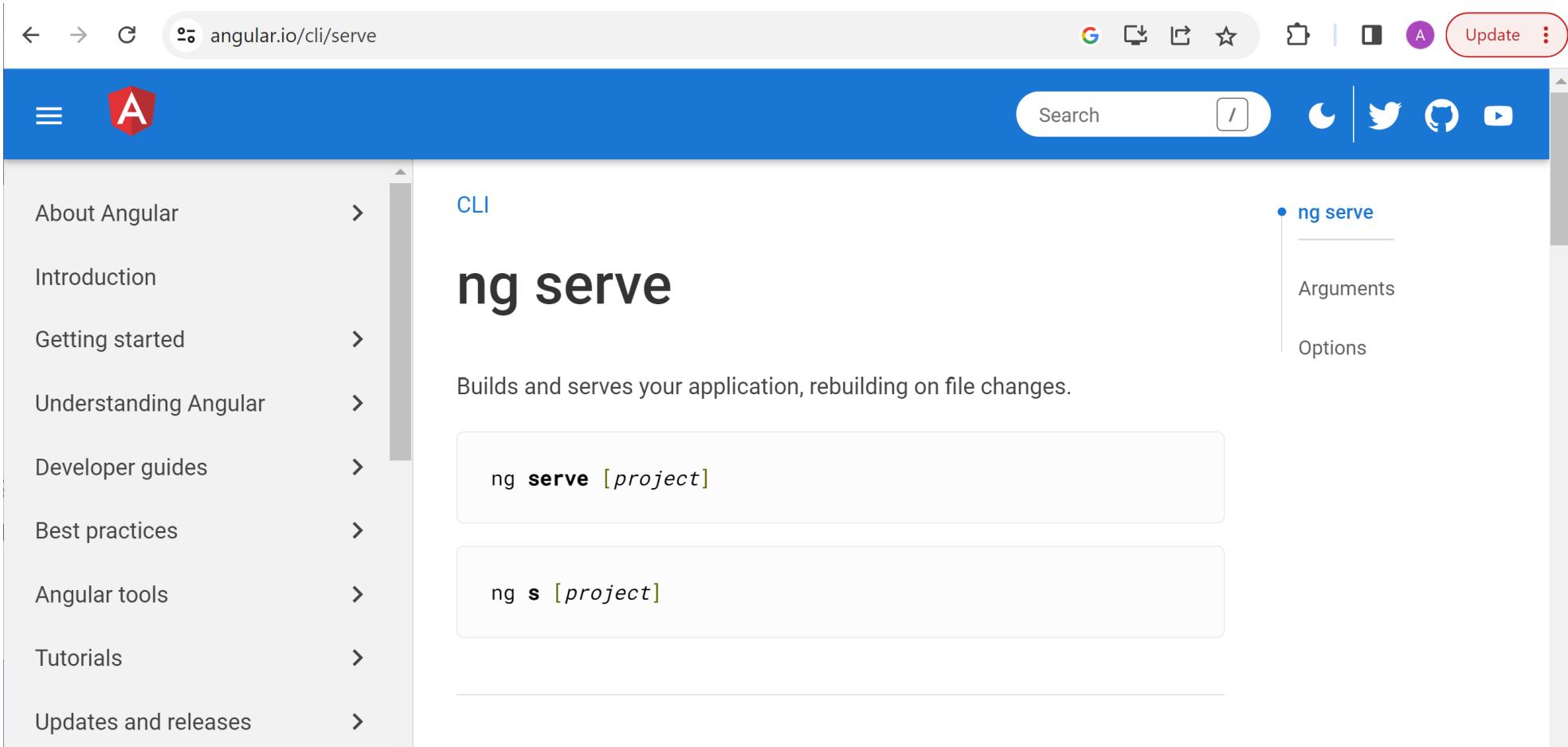
« Ng serve » has 1 , 2 or 3 roles !

Role 1 : basic development: serve **static local files**  
(\*.html, \*.js, \*.css ... compiled from \*.html, \*.ts, \*.scss )

Role 2 : for unit testing « **mock server** »  
requests in \*.json /api/\* in app/server/\*.ts

Role 3 : **proxy to backend** (eg. java)  
requests in \*.json /api/\* redirected to backend api

<https://angular.io/cli/serve>



The screenshot shows a web browser displaying the Angular CLI documentation for the `ng serve` command. The URL in the address bar is `https://angular.io/cli/serve`. The page has a blue header with the Angular logo and navigation links for About Angular, Introduction, Getting started, Understanding Angular, Developer guides, Best practices, Angular tools, Tutorials, and Updates and releases. The main content area is titled "CLI" and focuses on the `ng serve` command. It explains that it builds and serves your application, rebuilding on file changes. Two command examples are shown in boxes: `ng serve [project]` and `ng s [project]`. A sidebar on the right lists "ng serve" under "Arguments" and "Options". The browser interface includes a search bar, social sharing icons, and a red "Update" button.

About Angular

Introduction

Getting started

Understanding Angular

Developer guides

Best practices

Angular tools

Tutorials

Updates and releases

CLI

# ng serve

Builds and serves your application, rebuilding on file changes.

```
ng serve [project]
```

```
ng s [project]
```

ng serve

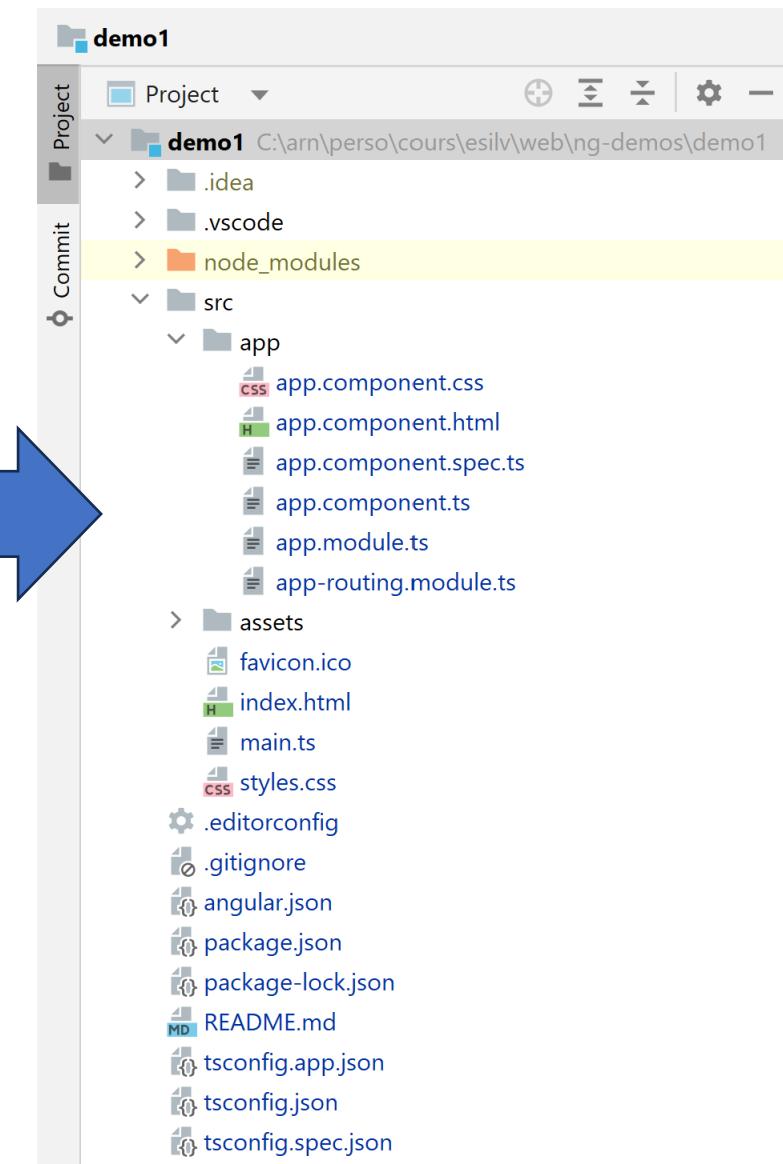
Arguments

Options

# (pre-requisite) npm install -g @angular/cli

## ng new

```
$ ng new  
? What name would you like to use for the new workspace and initial project? demo1  
? Would you like to add Angular routing? Yes  
? Which stylesheet format would you like to use? CSS  
CREATE demo1/angular.json (2695 bytes)  
CREATE demo1/package.json (1036 bytes)  
CREATE demo1/README.md (1059 bytes)  
CREATE demo1/tsconfig.json (901 bytes)  
CREATE demo1/.editorconfig (274 bytes)  
CREATE demo1/.gitignore (548 bytes)  
CREATE demo1/tsconfig.app.json (263 bytes)  
CREATE demo1/tsconfig.spec.json (273 bytes)  
CREATE demo1/.vscode/extensions.json (130 bytes)  
CREATE demo1/.vscode/launch.json (474 bytes)  
CREATE demo1/.vscode/tasks.json (938 bytes)  
CREATE demo1/src/favicon.ico (948 bytes)  
CREATE demo1/src/index.html (291 bytes)  
CREATE demo1/src/main.ts (214 bytes)  
CREATE demo1/src/styles.css (80 bytes)
```



ng serve => http://localhost:4200  
... live-reload on \*.html, \*.ts

```
$ ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files      | Names          | Raw Size
vendor.js                | vendor        | 2.04 MB
polyfills.js              | polyfills    | 314.82 kB
styles.css, styles.js     | styles       | 209.94 kB
main.js                   | main         | 48.08 kB
runtime.js                | runtime      | 6.51 kB

| Initial Total | 2.61 MB

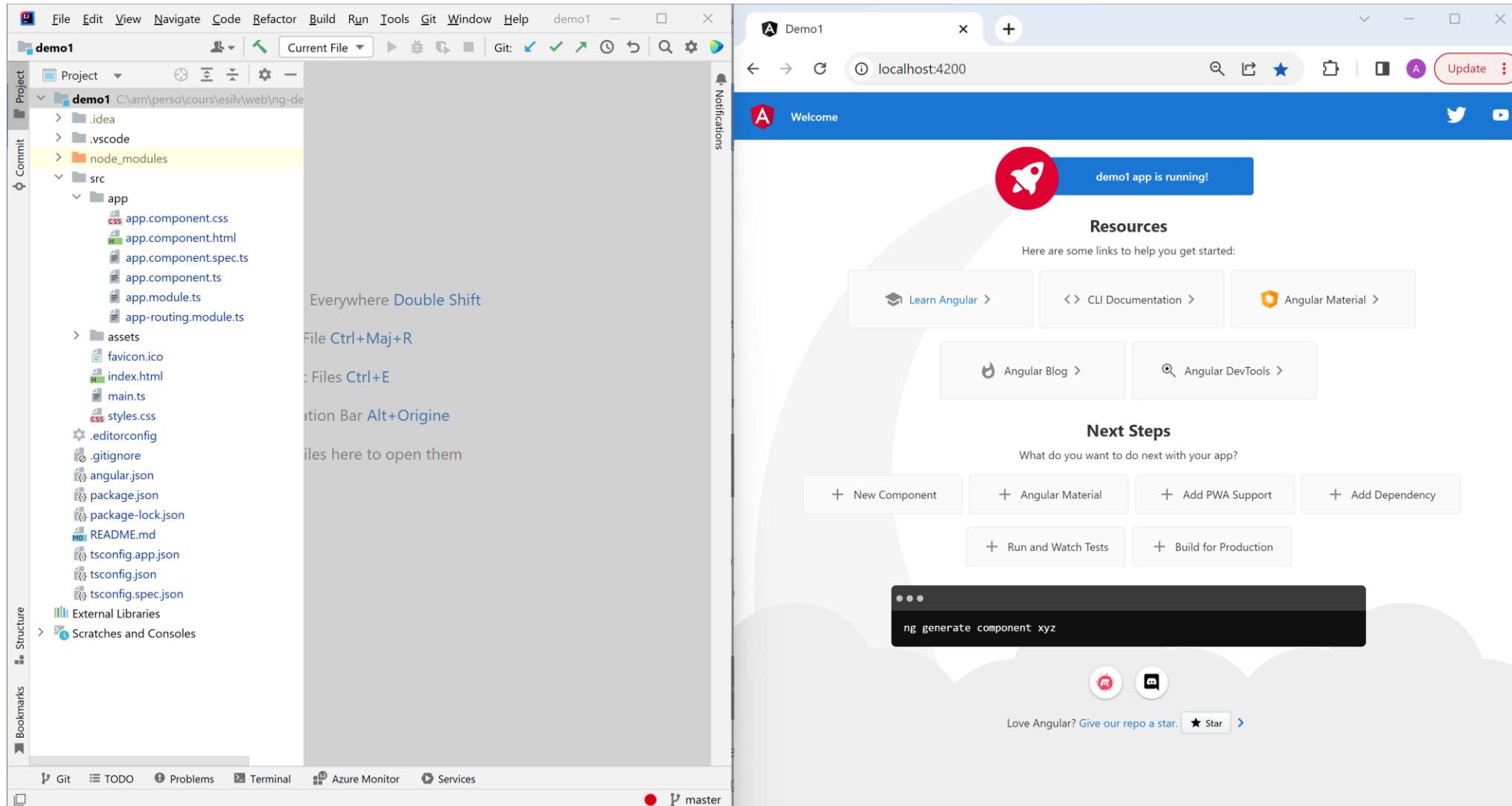
Build at: 2023-07-09T10:06:15.879Z - Hash: 70e19beef187b407 - Time: 22482ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
```

# Demo

## IntelliJ (left) + Chrome (right) + ng serve



# Demo ... Live reload

The screenshot shows a development environment with an IDE on the left and a browser window on the right.

**IDE (Left):**

- Project:** demo1 (C:\arn\perso\cours\esilv\web\ng-de)
- Current File:** app.component.html
- Code Editor:** Content of app.component.html:

```
live-reload Test !!  
<router-outlet></router-outlet>
```
- Notifications:** A yellow warning icon with the number 2 is visible.
- Toolbars:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help.
- Bottom:** Git, TODO, Problems, Terminal, Azure Monitor, Services, Status bar (2:20, LF, UTF-8, 2 spaces\*, master).

**Browser (Right):**

- Title:** Demo1
- Address:** localhost:4200
- Content:** live-reload Test !!
- Toolbar:** Back, Forward, Refresh, Stop, Home, Search, Favorites, Update, More.

# Live Reload ... --watch and compile !!

```
  Cmder
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names      | Raw Size
runtime.js          | runtime    | 6.51 kB |
main.js             | main       | 5.95 kB |

3 unchanged chunks

Build at: 2023-07-09T10:12:32.466Z - Hash: cad54614d11ba0ea - Time: 727ms

✓ Compiled successfully.
✓ Browser application bundle generation complete.

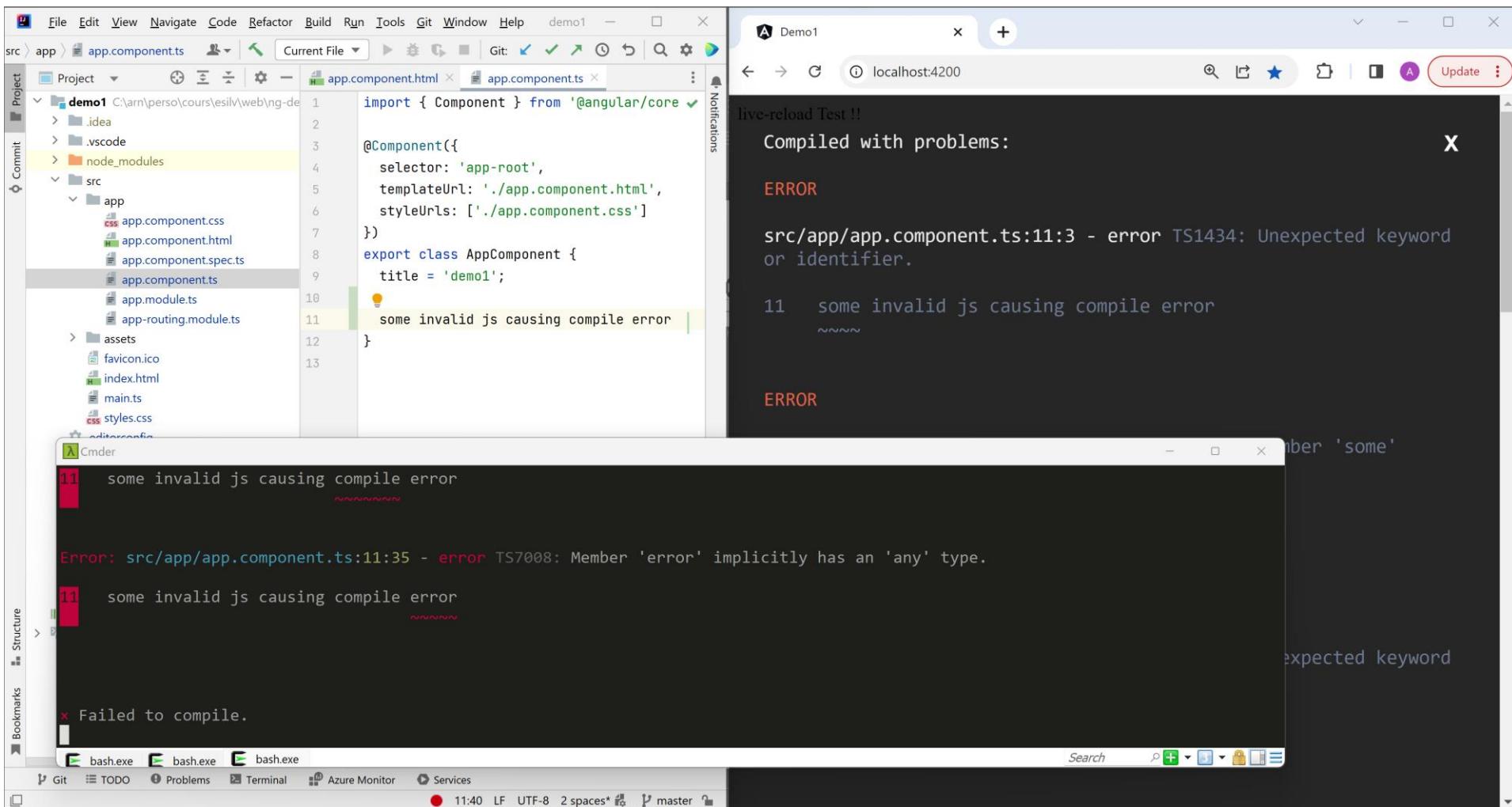
Initial Chunk Files | Names      | Raw Size
runtime.js          | runtime    | 6.51 kB |
main.js             | main       | 5.95 kB |

3 unchanged chunks

Build at: 2023-07-09T10:13:12.476Z - Hash: 00615d4add0fa892 - Time: 298ms

✓ Compiled successfully.
```

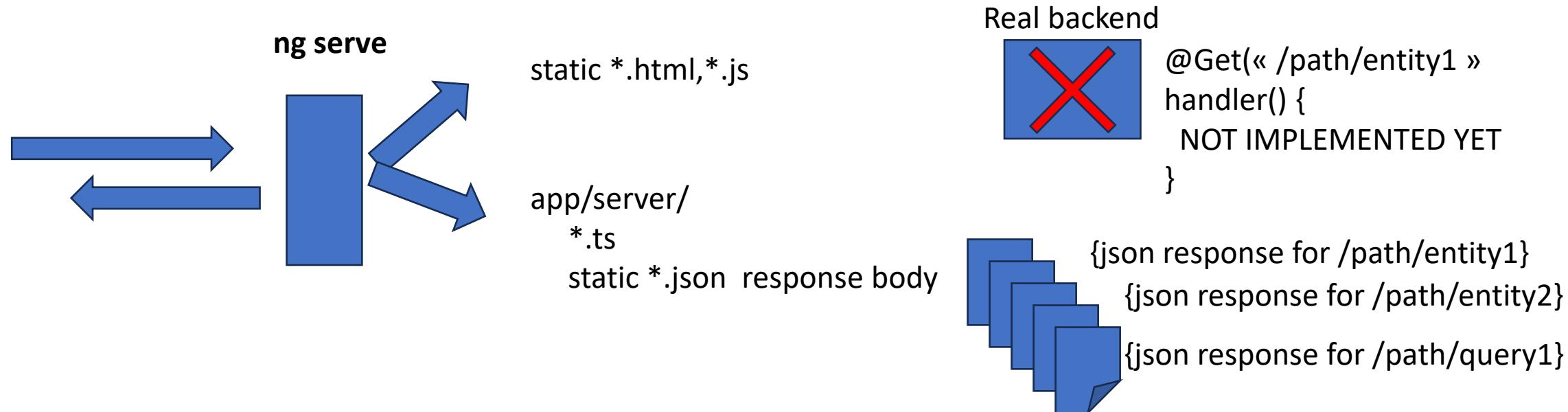
# Demo live-reload ... when compile error



# Ng serve... why Mock « app/server/\*ts, \*.json »

Because ...

Web developper/designer can start working in parallel without backend !!

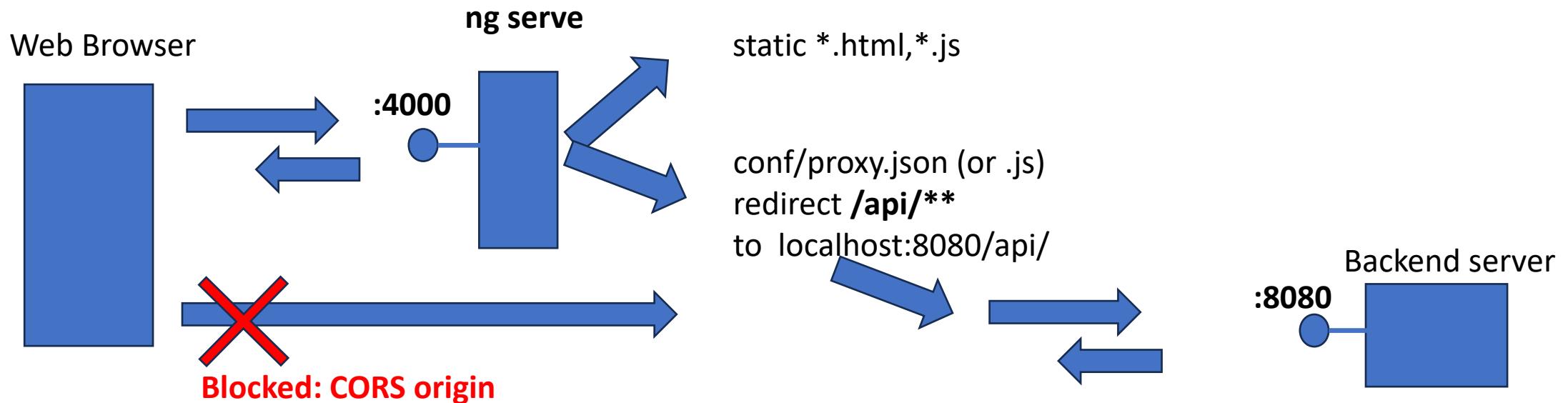


# Ng serve... why Proxy to /api/\*\* ? => CORS Origin problem !

Because ...

Web Browser blocks calls to « http://localhost:**8080**/api/\*\* »

not comming from same « origin domain » as « http://localhost:**4000**/index.html » «app.js » !



# Virtual Host ... same IP address, different « Virtual » Server Hosts

\$ nslookup virtual-host1  
=> sharedIP

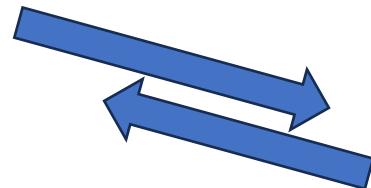
\$ nslookup virtual-host2  
=> sharedIP (same!!)

DNS server

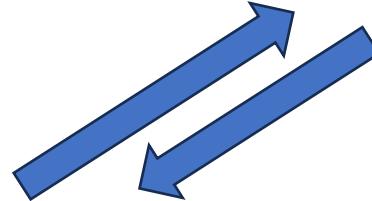


DNS entry1: virtual-host1=sharedIP  
DNS entry2: virtual-host2=sharedIP

http {verb} /{path}  
Headers...  
**« host=virtual-host1 »**

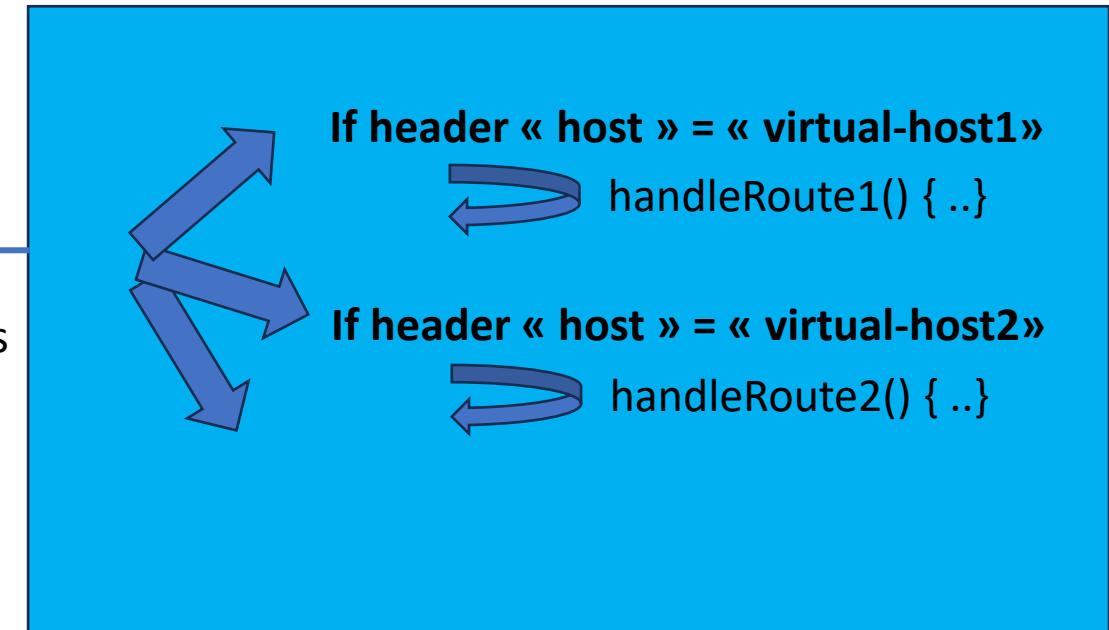


http {verb} /{path}  
Headers...  
**« host=virtual-host2 »**



1 IP address  
ONLY !!!

**HTTP Server**  
(single IP address ... several virtual hosts)



Examples:

- Apache virtual host
- K8s ingress host

# Proxy Server

## Proxy server

54 languages ▾

Contents [\[hide\]](#)

Article [Talk](#)

Read [Edit](#) [View history](#) [Tools](#) ▾

(Top)

> [Types](#)

> [Uses](#)

> [Implementations of proxies](#)

See also

References

External links

From Wikipedia, the free encyclopedia

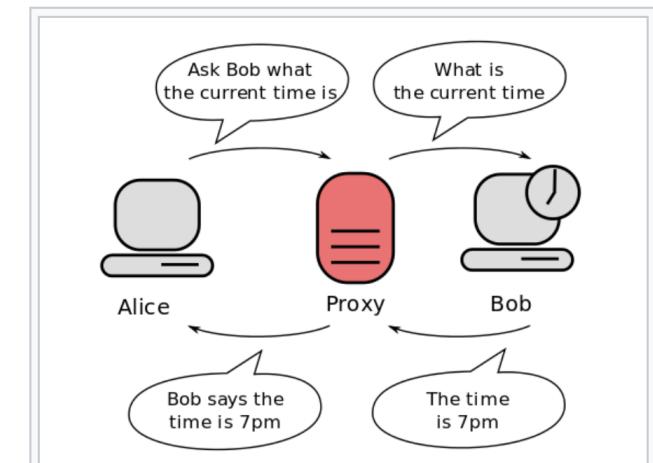
*For Wikipedia's policy on editing from open proxies, please see [Wikipedia:Open proxies](#). For other uses, see [Proxy](#).*

In [computer networking](#), a **proxy server** is a [server application](#) that acts as an [intermediary](#) between a [client](#) requesting a [resource](#) and the server providing that resource.<sup>[1]</sup> It improves privacy, security, and performance in the process.

Instead of connecting directly to a server that can fulfill a request for a resource, such as a file or [web page](#), the client directs the request to the proxy server, which evaluates the request and performs the required network transactions. This serves as a method to simplify or control the complexity of the request, or provide additional benefits such as [load balancing](#), privacy, or security. Proxies were devised to add structure and [encapsulation](#) to [distributed systems](#).<sup>[2]</sup> A proxy server thus functions on behalf of the client when requesting service, potentially masking the true origin of the request to the resource server.

### Types [\[edit\]](#)

A proxy server may reside on the user's [local computer](#), or at any point between



Communication between two computers connected through a third computer acting as a proxy server. This can protect Alice's privacy, as Bob only knows about the proxy and cannot identify or contact Alice directly.

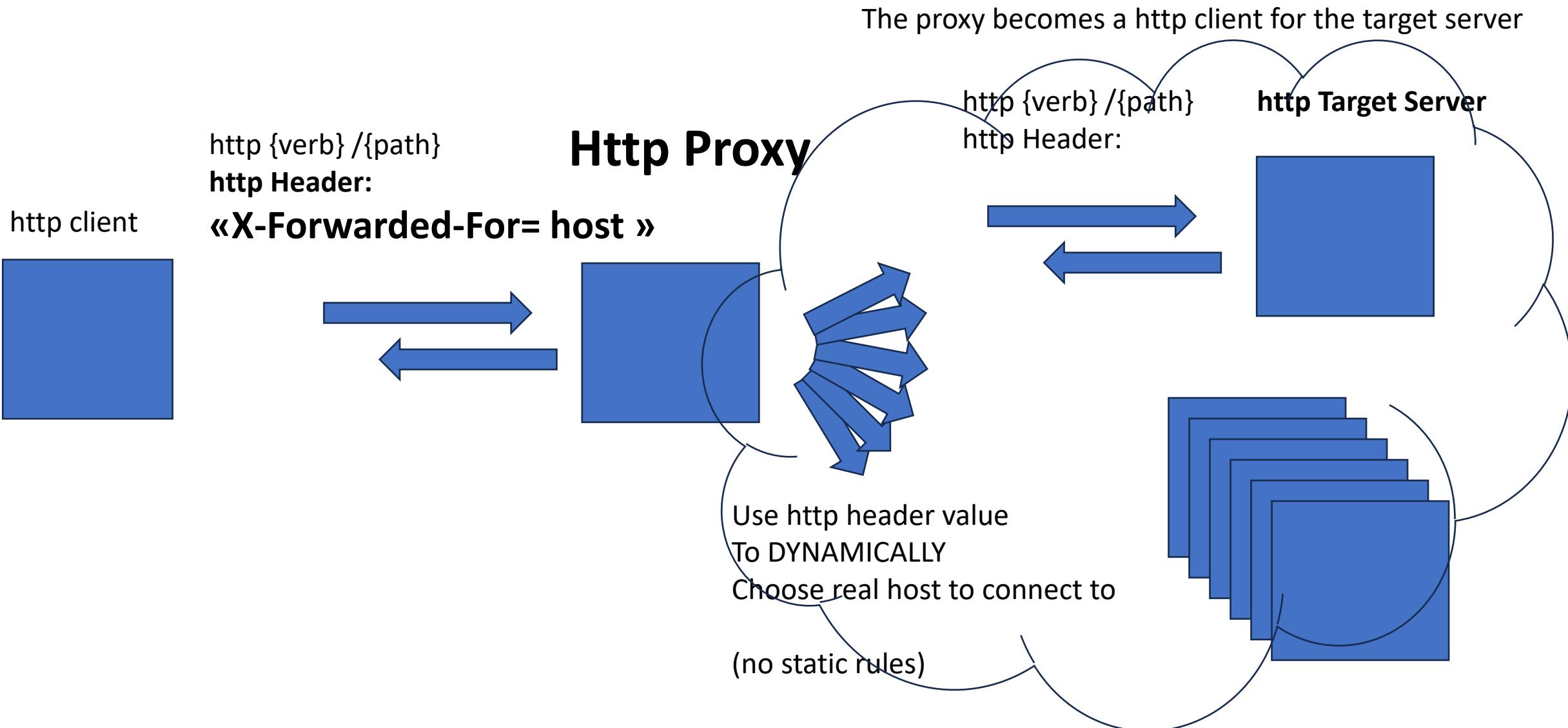
# Forward Proxy ... Reverse Proxy ??

Some Terminology...

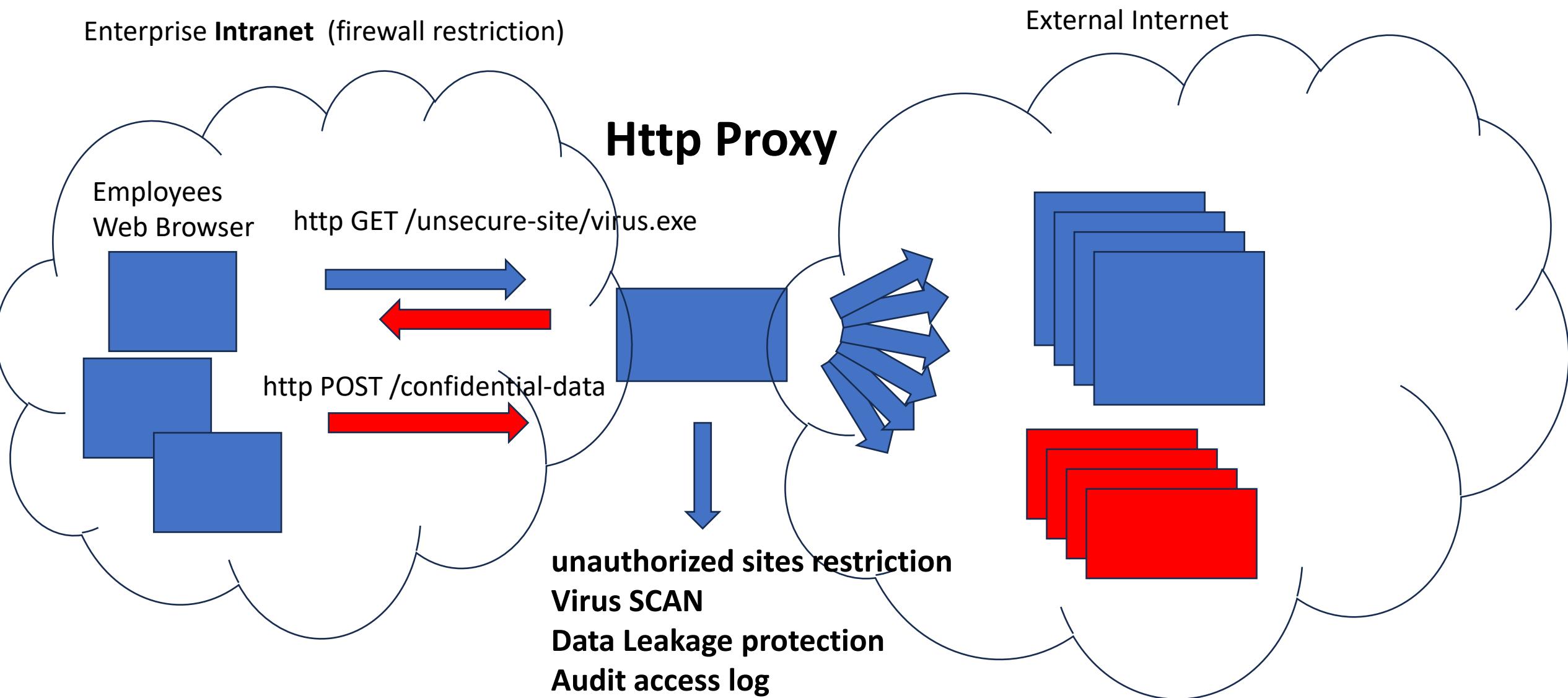
« **Reverse Proxy** » = **transparent** proxy = Gateway  
looks like a standard server  
( not seen as a http or socks proxy)  
Example: Used for loadBalancing, ingress in K8s

« **Forward** » proxy:  
use explicit « **X-Forwarded-For** » **http header**  
for intranet security + firewall + authentication..

# Http Proxy



# Example Intranet http « Security Proxy »



# Using « Transparently » a http Proxy

```
$ export HTTP_PROXY=http://proxyhost:8080
```

```
$ curl http://www.google.fr
```

=> use TCP-IP connection to « proxyhost » for all queries,  
And rewrite http request to add header « proxy=www.google.fr»

HTTP\_PROXY is standard environment variable

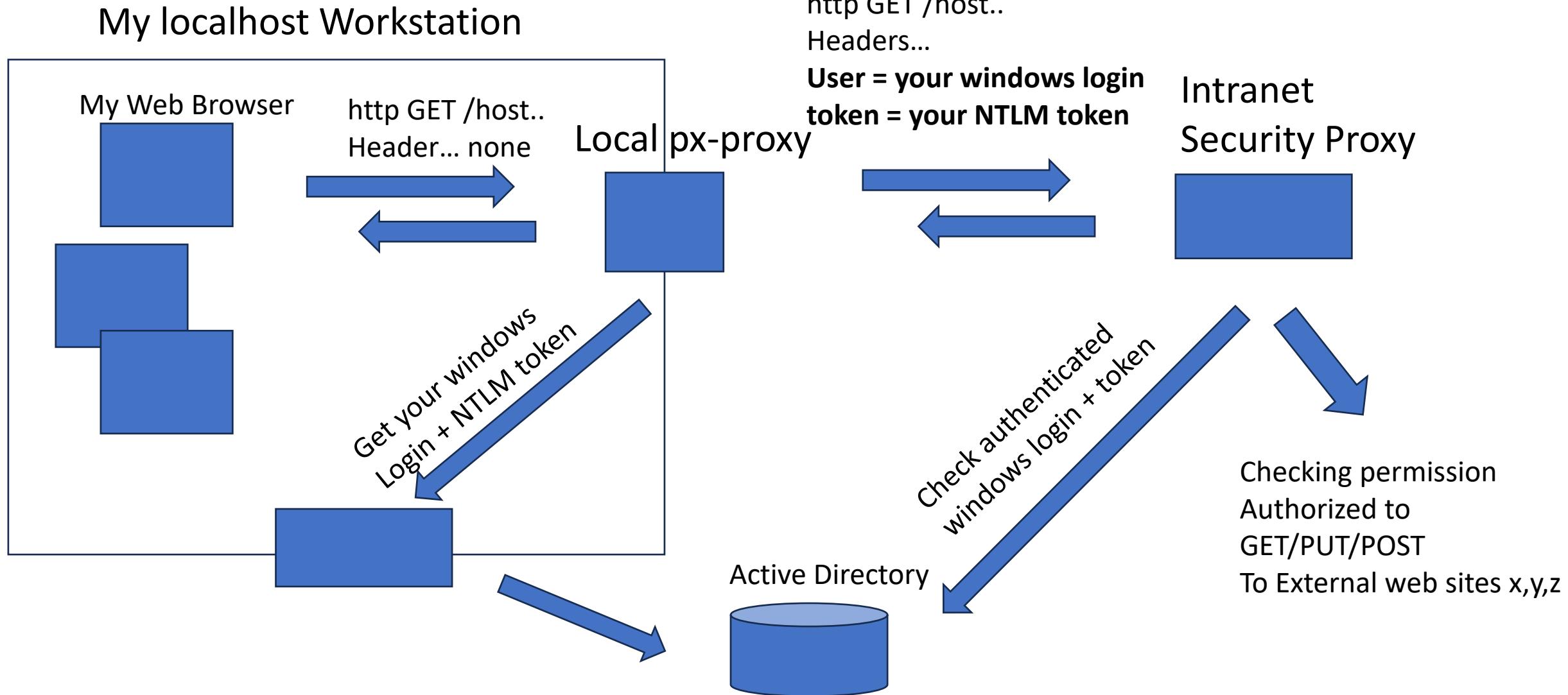
There are also standard configuration file for many applications (example: <proxy>..</proxy> in maven settings.xml  
And custom args per langage, example « java -Djava.net.proxy.http=proxyhost your-app.jar »

# Using « Transparently » an http request enricher Proxy ... authenticating

```
$ export HTTP_PROXY=localhost:3128
```

=> typically to use NTLM authenticating local proxy  
... to connect to enterprise Internet proxy  
Using your enterprise Windows account + NTLM token

# Px-proxy



# Summary

http protocol = Simple Text protocol + extensible Headers on top of TCP-IP

Few standards and norms: status code, header, cors origin

Types of servers:

- Static pages server
- Dynamic server-side pages (legacy)
- Api server (Rest Json)
- Mixed content : Gateway (Ingress Rules), Virtual Hosts
- Proxies
- Local development server: ng serve

# Next course / Hands-on

## Pre-requisite to install on your PCs:

### IDE

Higly recommended :  
**WebStorm** (or Visual Studio Code)

### Runtime

**Nodejs**  
**(+ npm)**

### Tool

**@angular-cli**  
**(npm module in PATH)**

WebStorm = JetBrains ~IntelliJ for Web  
also Included in « IntelliJ Ultimate »  
But not in « IntelliJ Community »

# Pre-Requisite Install 1/3 : WebStorm 30 days free trial

<https://www.jetbrains.com/idea/download>

The screenshot shows a web browser window displaying the JetBrains website for downloading WebStorm. The URL in the address bar is [jetbrains.com/webstorm/download/#section=windows](https://www.jetbrains.com/webstorm/download/#section=windows). The page features the JetBrain's logo and navigation links for Developer Tools, Team Tools, Education, Solutions, Support, and Store. Below the navigation bar, there are links for What's New, Features, Learn, Pricing (which is highlighted), and Download. A large 'WebStorm' logo is on the left, and a 'Download WebStorm' section is on the right. This section includes links for Windows, macOS, and Linux, and a note about a 30-day trial. A 'Download' button is available, along with a dropdown menu for file formats (.exe). At the bottom, there's a callout for the Toolbox App.

Download WebStorm

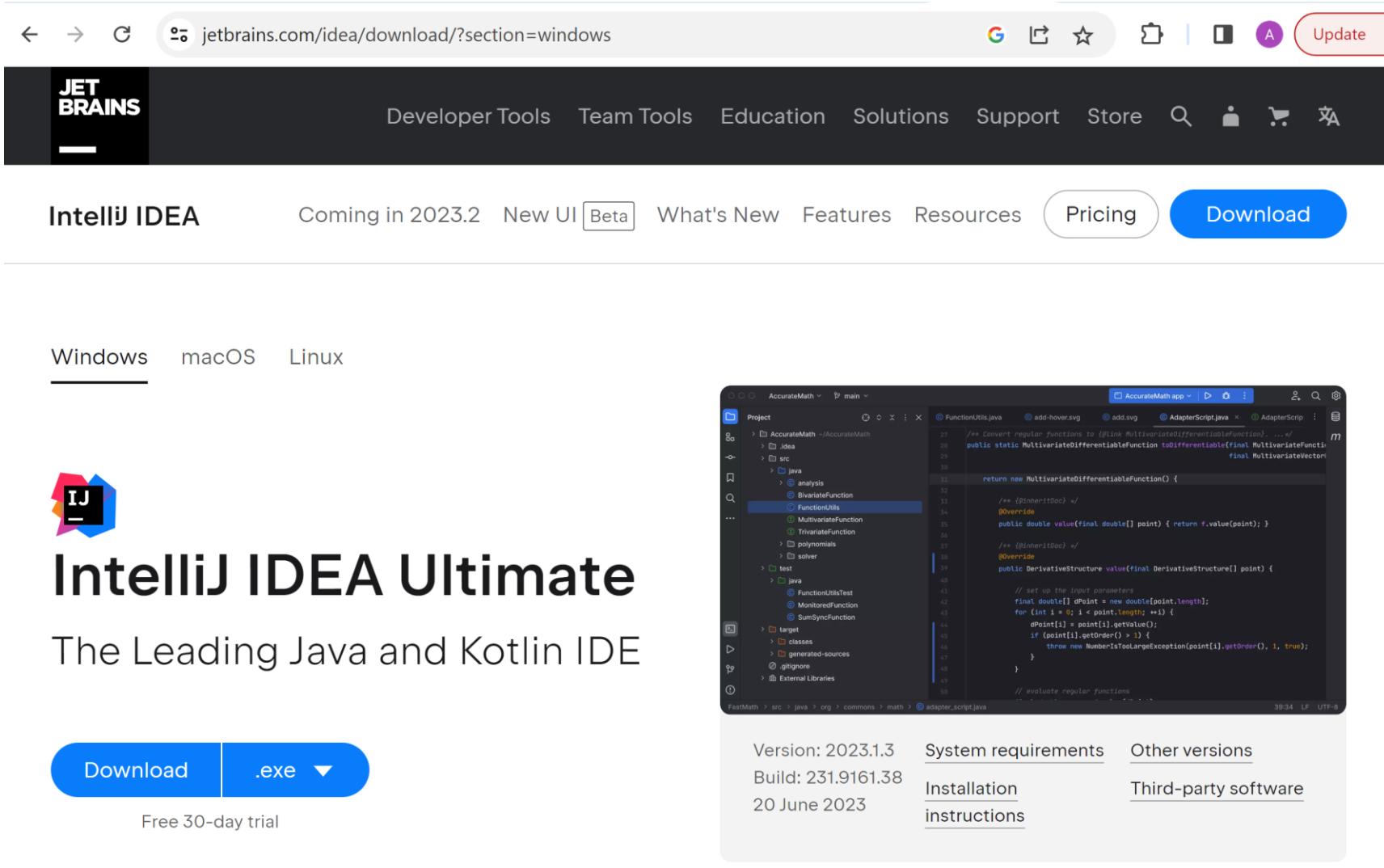
Windows macOS Linux

WebStorm includes an evaluation license key for a **free 30-day trial**.

Download .exe ▾

Get the Toolbox App to download WebStorm and its future updates with ease

(( OR equivalent ... IntelliJ Ultimate 30 days Free trial  
<https://www.jetbrains.com/idea/download/> )))



The screenshot shows the IntelliJ IDEA Ultimate download page on the JetBrains website. At the top, there's a navigation bar with links for Developer Tools, Team Tools, Education, Solutions, Support, Store, and a search icon. Below the navigation bar, there are tabs for IntelliJ IDEA, Coming in 2023.2, New UI (Beta), What's New, Features, Resources, Pricing (which is highlighted in a blue box), and Download. The main content area features a large image of the IntelliJ IDE interface, showing code editor, project tree, and toolbars. Below the image, there are download links for Windows, macOS, and Linux. A prominent "IntelliJ IDEA Ultimate" logo with a stylized "IJ" icon is displayed. The text "The Leading Java and Kotlin IDE" is also present. At the bottom, there are download buttons for ".exe" and "Download", along with a "Free 30-day trial" link. The footer contains version information (Version: 2023.1.3, Build: 231.9161.38, 20 June 2023) and links for System requirements, Other versions, Installation instructions, and Third-party software.

jetbrains.com/idea/download/?section=windows

JET BRAINS

Developer Tools Team Tools Education Solutions Support Store

IntelliJ IDEA Coming in 2023.2 New UI Beta What's New Features Resources Pricing Download

Windows macOS Linux

**IntelliJ IDEA Ultimate**

The Leading Java and Kotlin IDE

Download .exe ▾

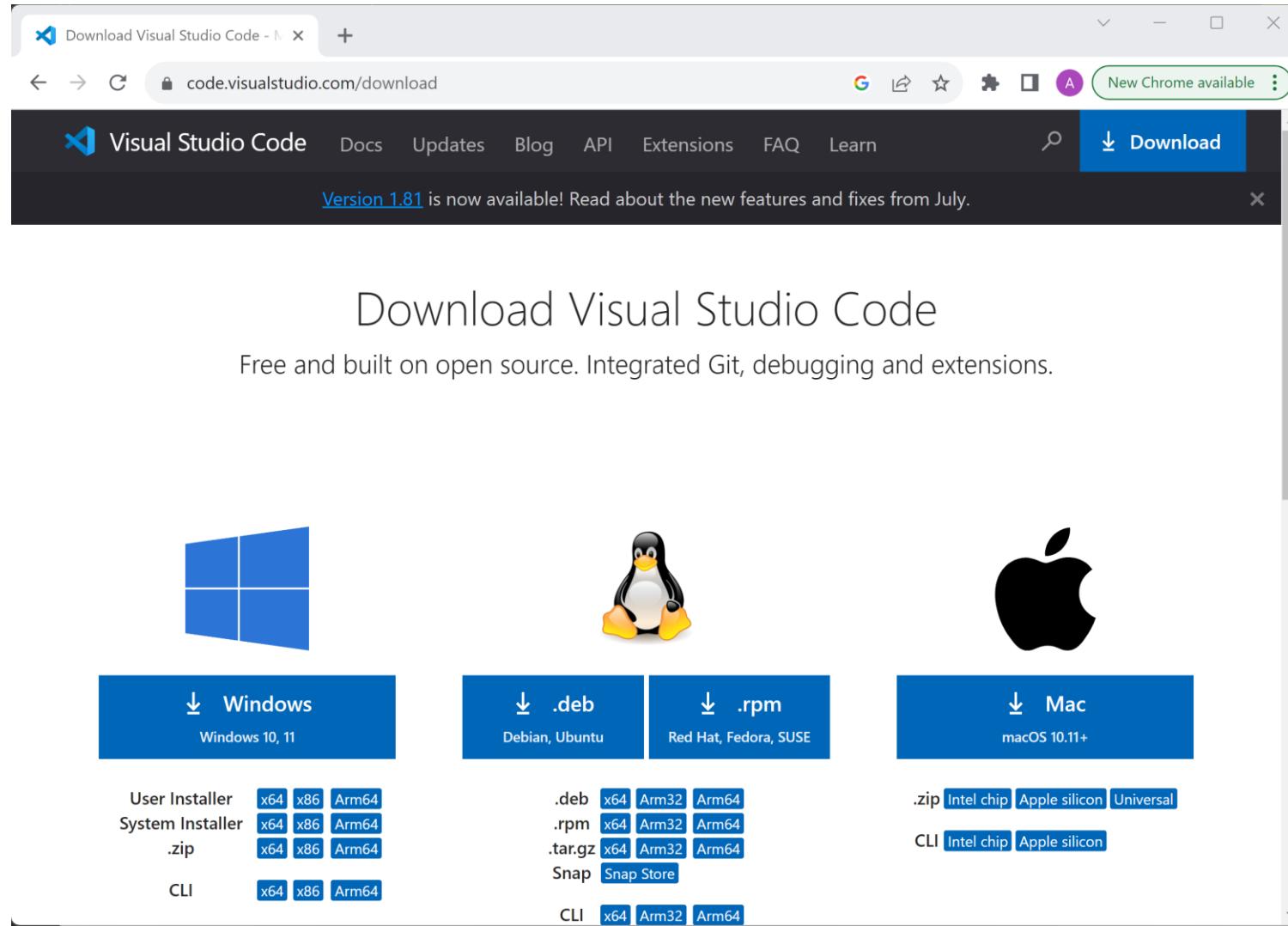
Free 30-day trial

Version: 2023.1.3  
Build: 231.9161.38  
20 June 2023

System requirements Other versions  
Installation instructions Third-party software

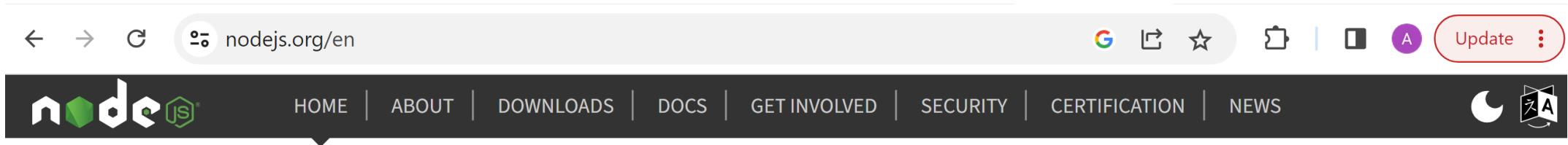
# OR ... Eclipse with \*Web plugins

# OR ... VisualStudio Code



# Pre-Requisite Install 2/3 : Nodejs

<https://nodejs.org>



Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download for Windows (x64)

18.16.1 LTS

Recommended For Most Users

20.4.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

# Pre-Requisite Install 3/3 : @angular/cli

The screenshot shows a web browser displaying the Angular documentation at [angular.io/guide/setup-local](https://angular.io/guide/setup-local). The page is titled "Setting up the local environment and workspace". On the left, there's a sidebar with navigation links like "About Angular", "Introduction", "Getting started" (which is expanded), "What is Angular?", "Try it", "Setup" (which is also expanded), "Understanding Angular", "Developer guides", "Best practices", "Angular tools", and "Tutorials". The main content area contains instructions for installing the Angular CLI with the command `npm install -g @angular/cli`. It also includes a note about setting execution policies on Windows. A sidebar on the right lists "Prerequisites", "Install the Angular CLI" (which is highlighted with a blue dot), "Create a workspace and initial application", "Run the application", and "Next steps".

To install the Angular CLI, open a terminal window and run the following command:

```
npm install -g @angular/cli
```

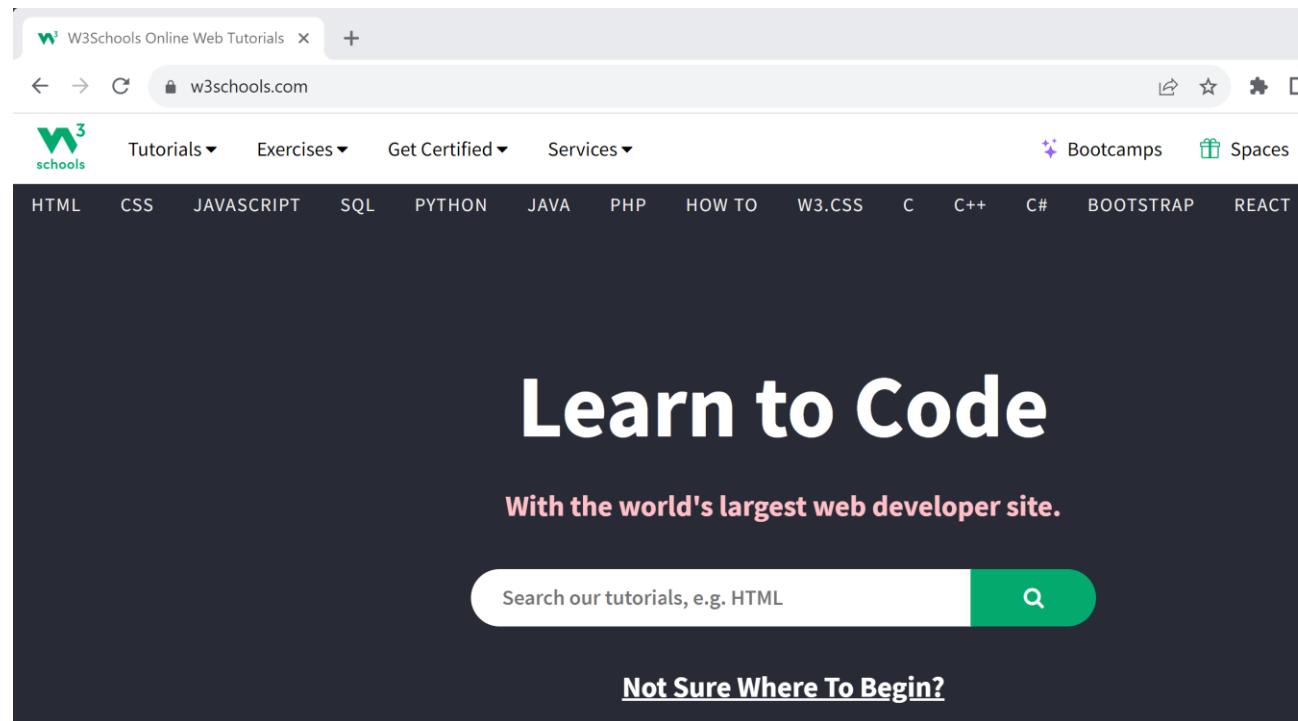
On Windows client computers, the execution of PowerShell scripts is disabled by default. To allow the execution of PowerShell scripts, which is needed for npm global binaries, you must set the following execution policy:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

# Suggested Reading for Next Time

<https://www.w3schools.com/>

**HTML + JavaScript + TypeScript + NodeJS + Angular ...**



# Suggested Reading for Next Time

<https://www.w3schools.com/js>

The screenshot shows the homepage of the JavaScript tutorial on w3schools.com. The URL in the address bar is `w3schools.com/js/default.asp`. The page title is "JavaScript Tutorial". A sidebar on the left lists various JavaScript topics like JS Introduction, JS Where To, JS Output, etc. The main content area contains introductory text about JavaScript being the world's most popular programming language and its use as the programming language of the Web. It also states that JavaScript is easy to learn and will teach from basic to advanced. A green button at the bottom left says "Start learning JavaScript now »".

<https://www.w3schools.com/nodejs>

The screenshot shows the homepage of the Node.js tutorial on w3schools.com. The URL in the address bar is `w3schools.com/nodejs/default.asp`. The page title is "Node.js Tutorial". A sidebar on the left lists various Node.js modules like Node.js Intro, Node.js Get Started, Node.js Modules, etc. The main content area contains introductory text about Node.js being an open source server environment and allowing you to run JavaScript on the server. A green button at the bottom right says "Start learning Node.js now »".

Questions ?

# MindMap to Take Away

<https://mm.tt/app/map/2853349794?t=AFOnMZf7Ku>



# Links

[arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

Course Esilv 2023

This document:

<https://github.com/Arnaud-Nauwynck/presentations/web/intro-web-dev-part1-http.pdf>