Introduction to Web Development

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

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»

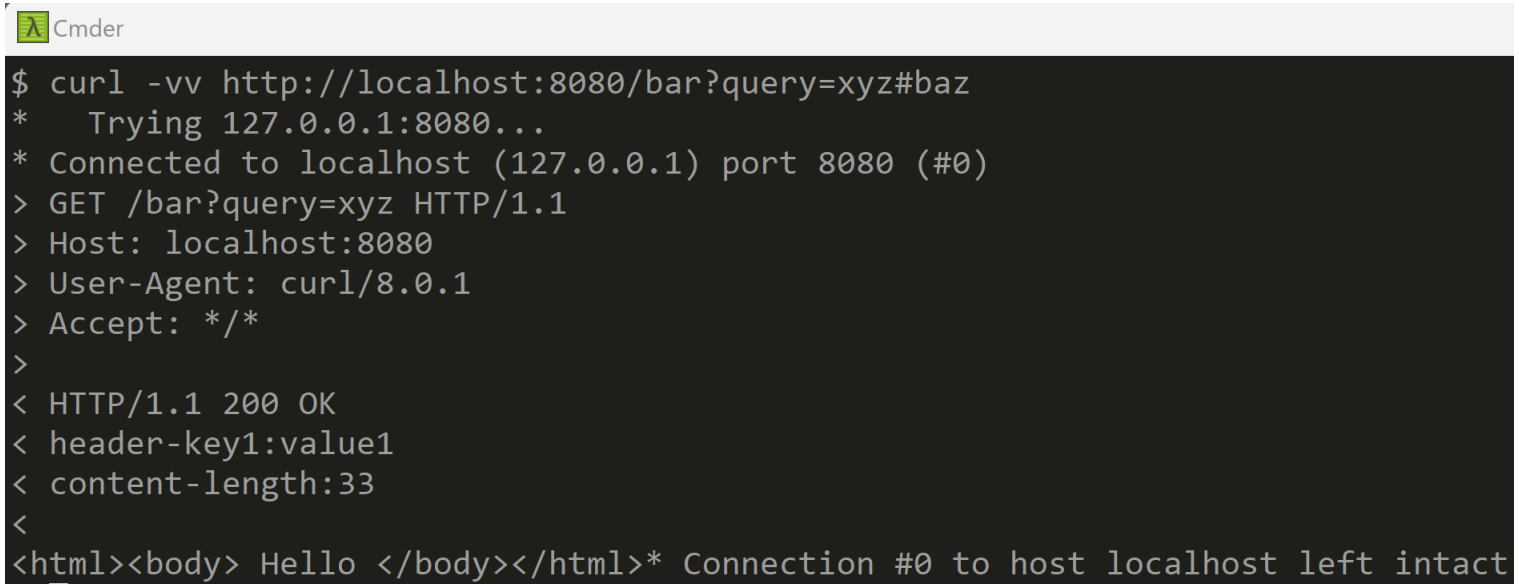
« http://localhost:8080/bar?query=xyz#baz»

??

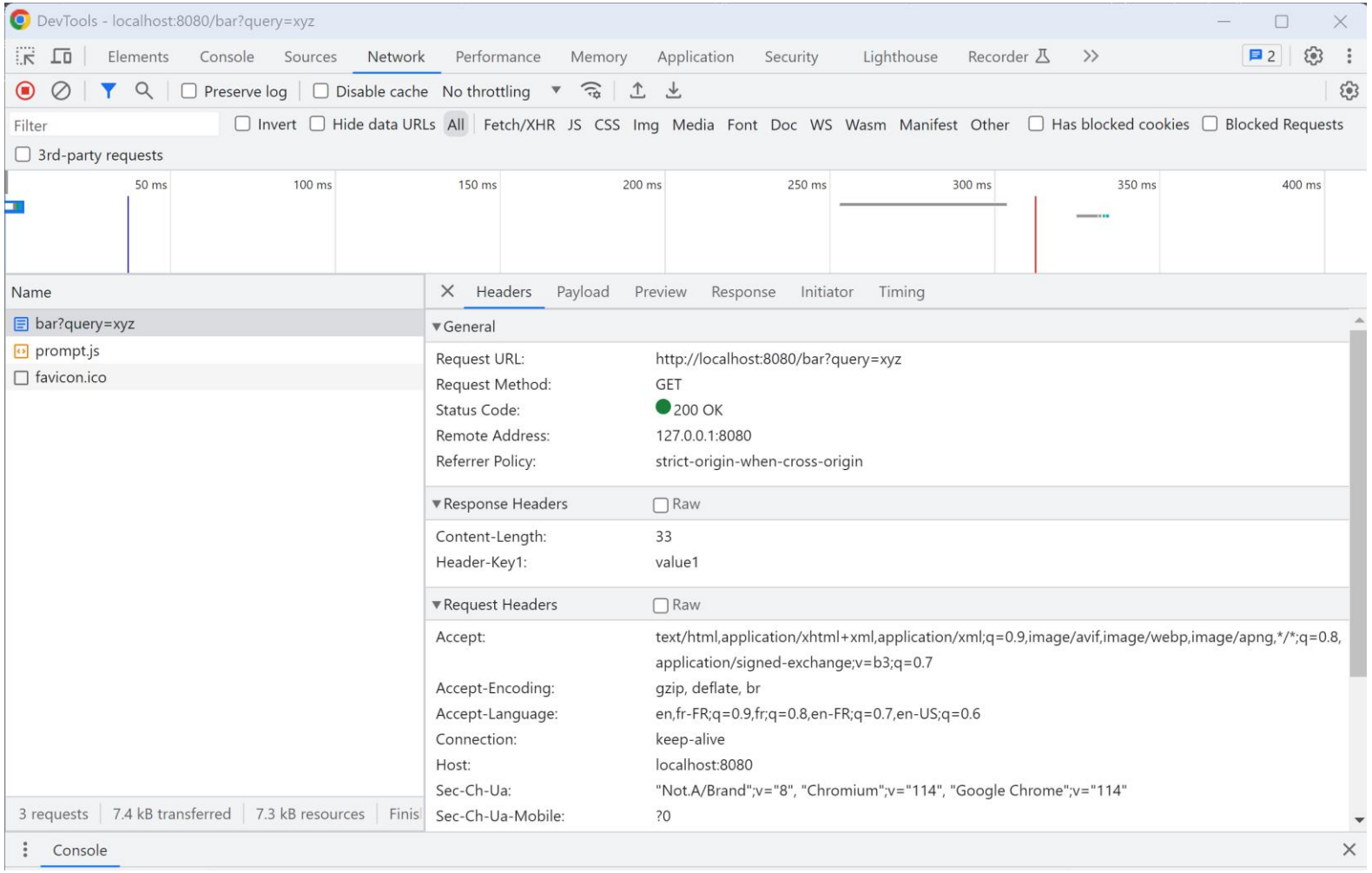
Guessed?
« http://localhost:8080» is NOT part of path
« #bar » is skipped

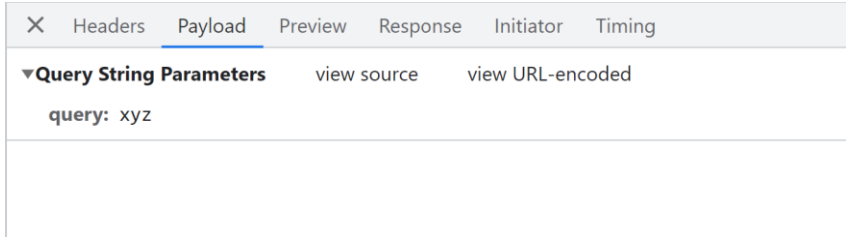
Interpreted because http is over TCP-IP

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

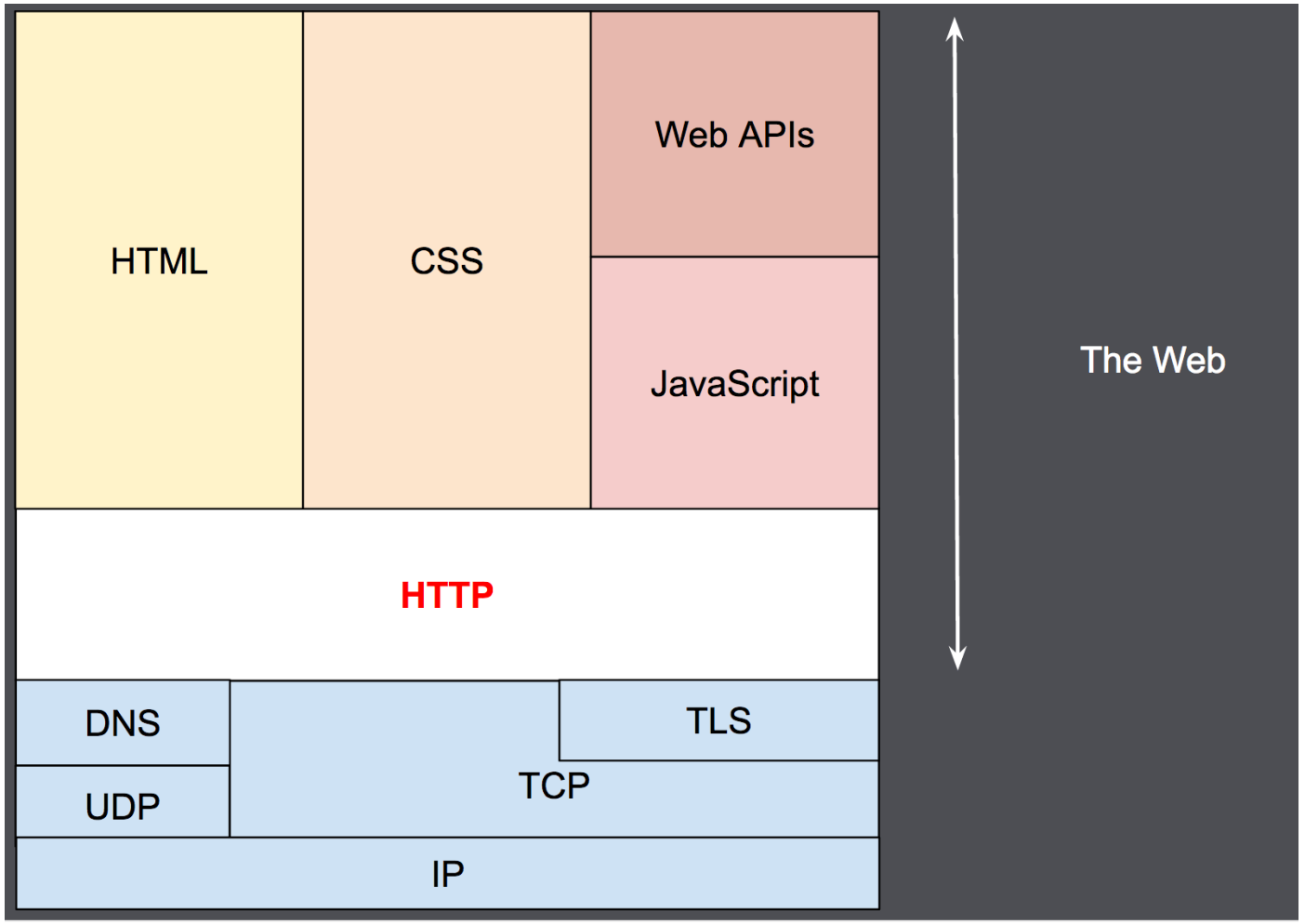


F12 : Chrome Debugger Tool > Network





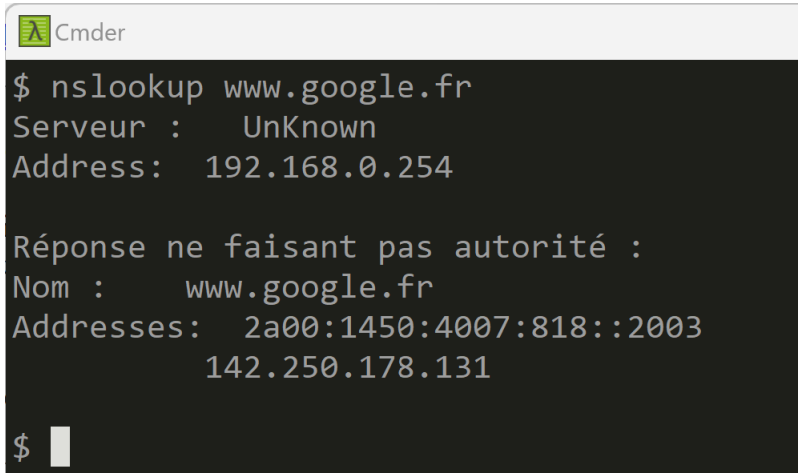
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview



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



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

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

Cmder

```
$ ipconfig

Suffixe DNS propre à la connexion. . . :
Adresse IPv6. . . . . . . . . . . . . .: 2a01:e0a:1f7:1940:9dd7:282d:539c:9657
Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:1f7:1940:7dac:dce3:2ff:dbfb
Adresse IPv6 de liaison locale. . . . .: fe80::9722:8385:d5a4:b6ce%15
Adresse IPv4. . . . . . . . . . . . . .: 192.168.0.10
Masque de sous-réseau. . . . . . . . . : 255.255.255.0
Passerelle par défaut. . . . . . . . . : fe80::6aa3:78ff:fe7e:3913%15
                                        192.168.0.254
```

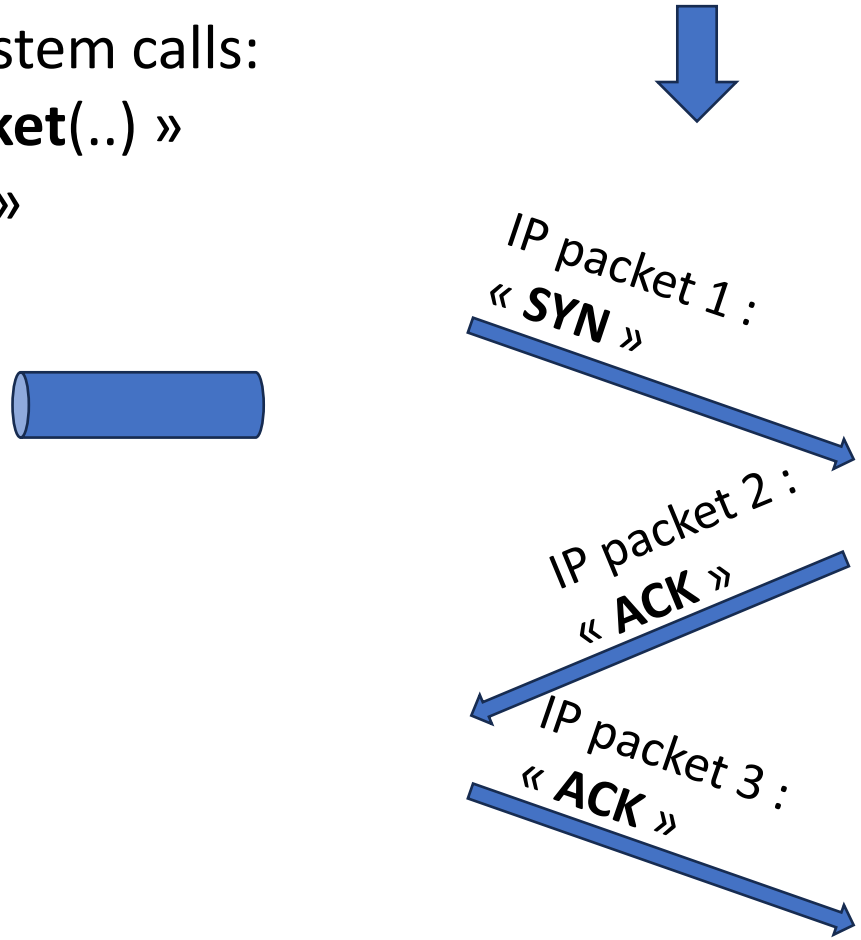
TCP-IP socket (=4 values in IP packets)

From IP: 192.168.0.10
fromPort:24352 (random)

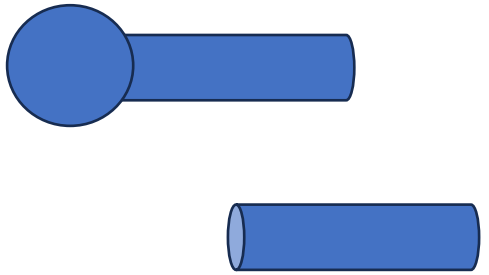
To IP : 142.250.178.131
toPort: 443

Step 3: TCP-IP socket connection

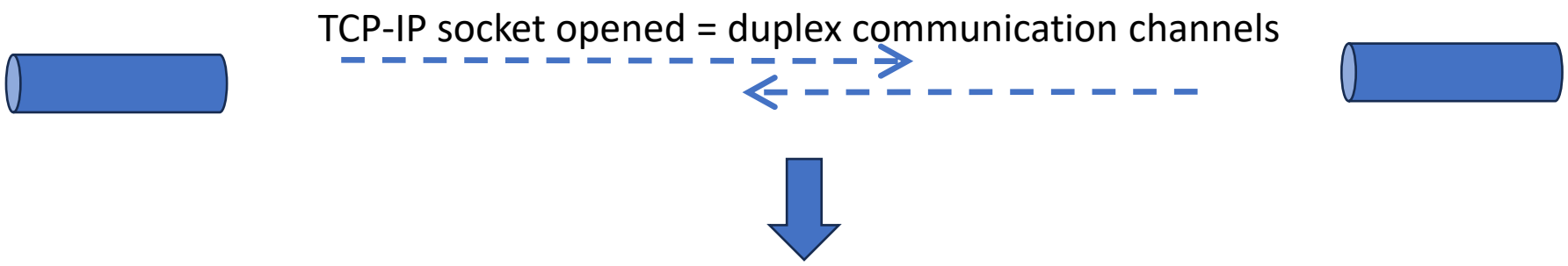
Client-Side System calls:
« s = new **socket**(..) »
« s.**connect**() »

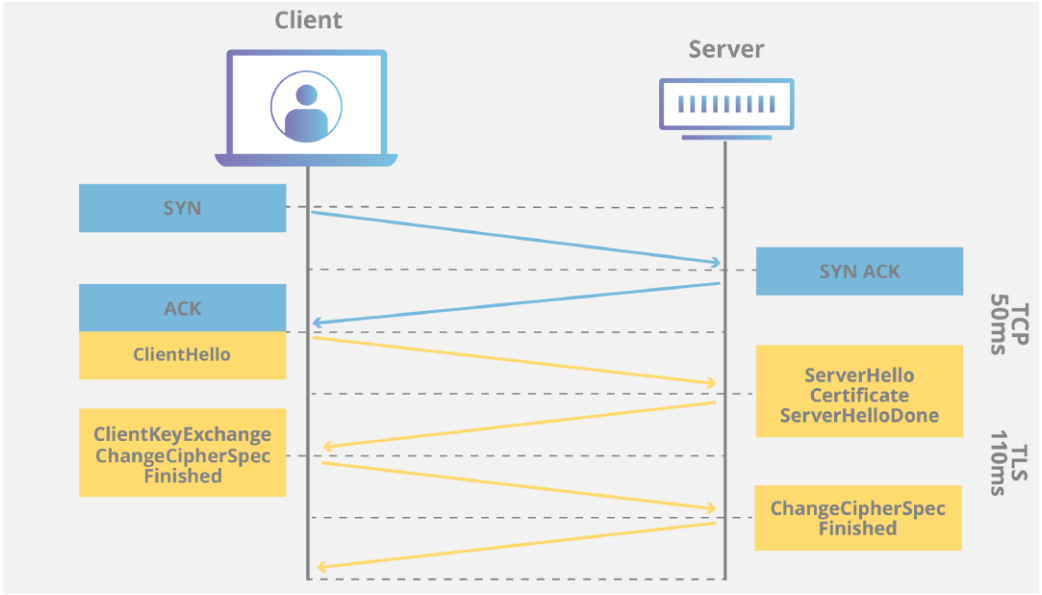


Server-Side System calls:
« ss = **new ServerSocket**() »
« ss.**bind**(loc,443); »
« s = ss.**accept**() »



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





Generate private+public key pair

check certificate

Send certificate
(=public key, signed by others)

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

Server WRITE response text,
then flush socket buffer

Line1 =
« HTTP/1.1 200 OK»

200,300,301,400,404,500,…
are http « **status code**»

« **status message**»

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

http Response Headers
some are standards headers: Content-Type, ..

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

http Response Body

Step 8: Client read response, close socket

http(s) Protocol Caracteristics

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

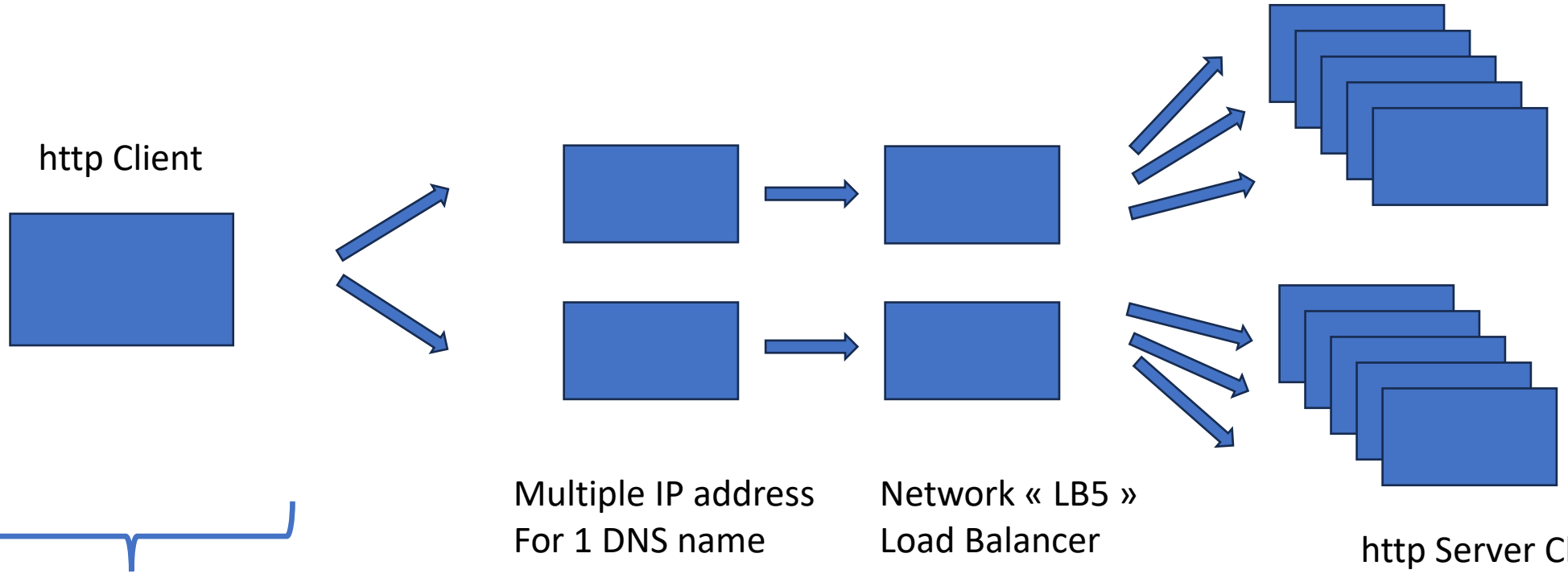
<body>

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 + horyzontal scalability)



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, langage 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

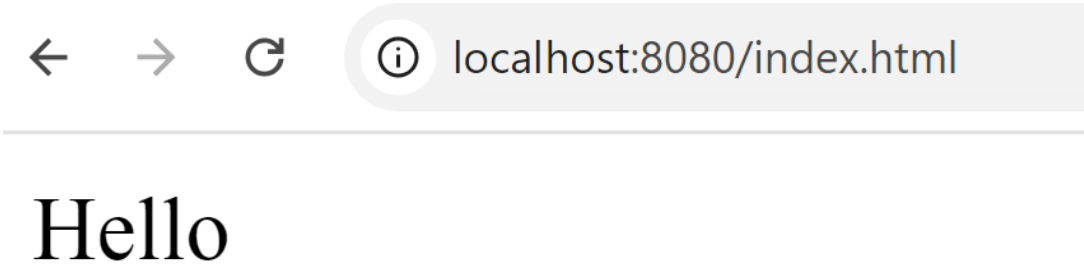
```
public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket();
    serverSocket.bind(new InetSocketAddress("localhost", 8080));
    for(;;) {
        Socket socket = serverSocket.accept();
        System.out.println("accepted socket connection from "
            + 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 …

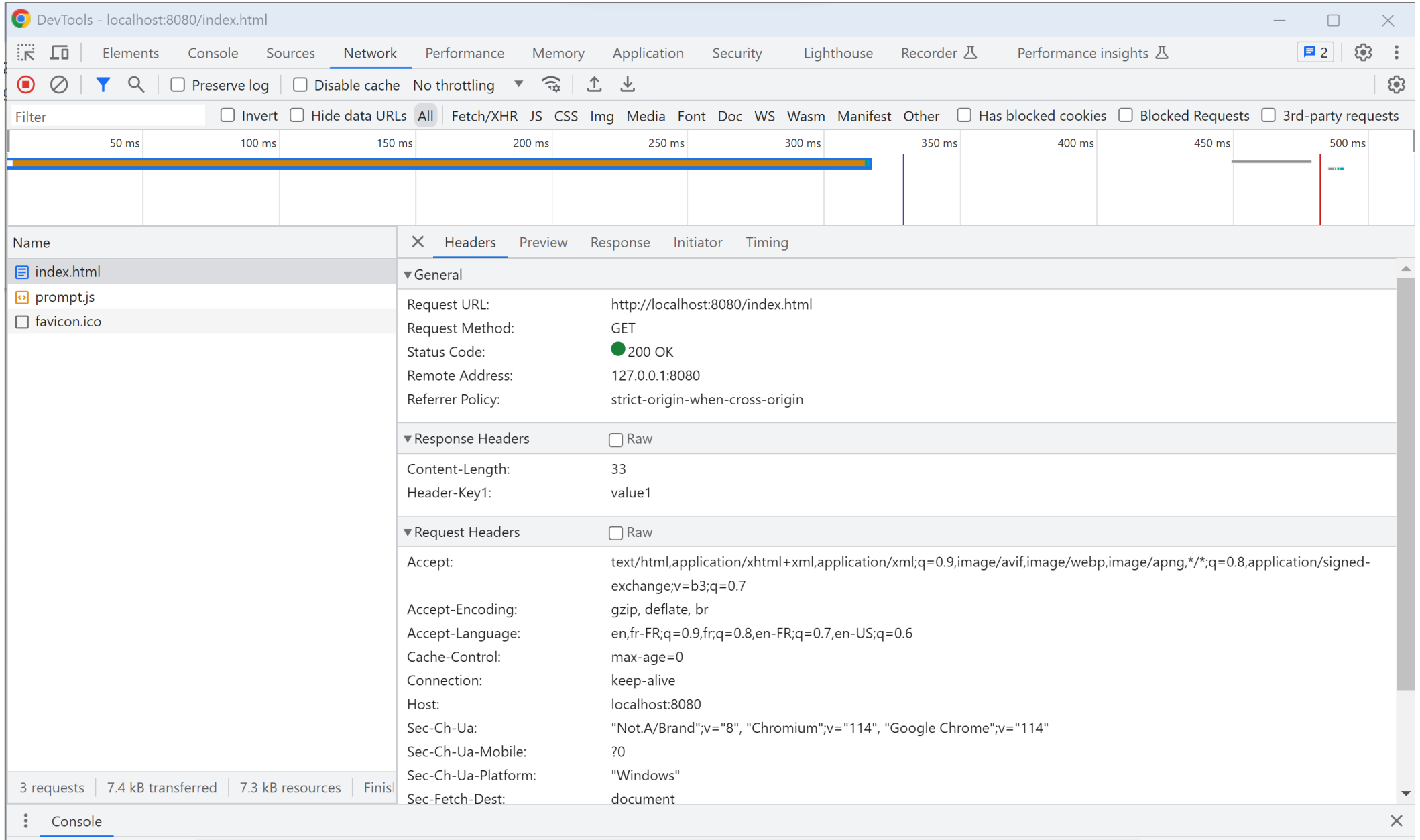
Launch Server

Open Chrome Web Browser

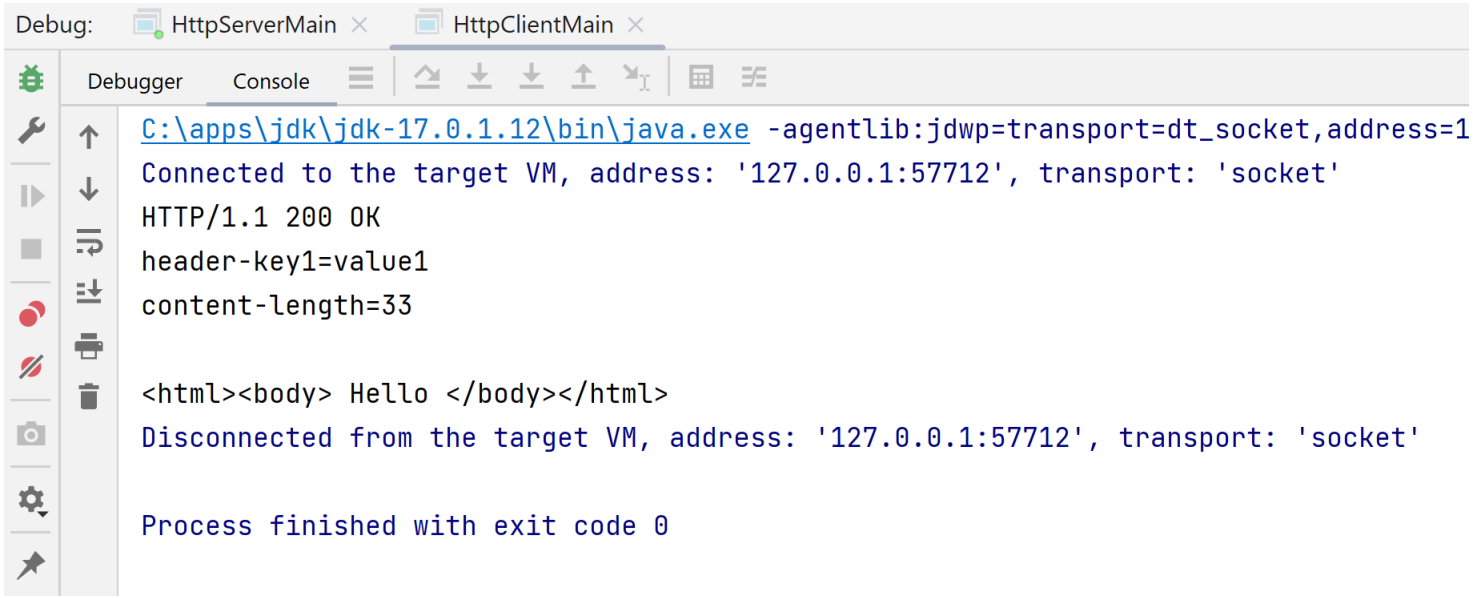
Type url: http://localhost:8080/

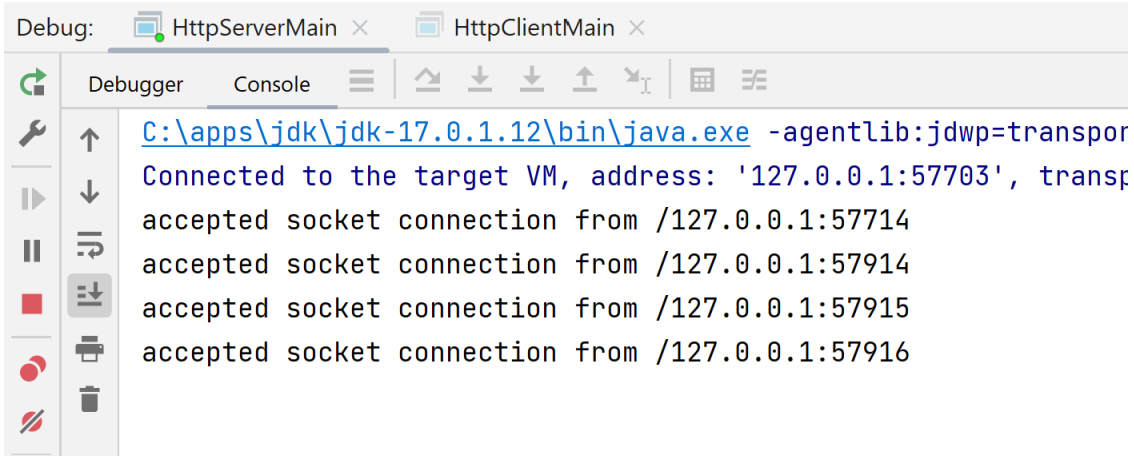


F12 (Chrome Debugger Tools > Network)

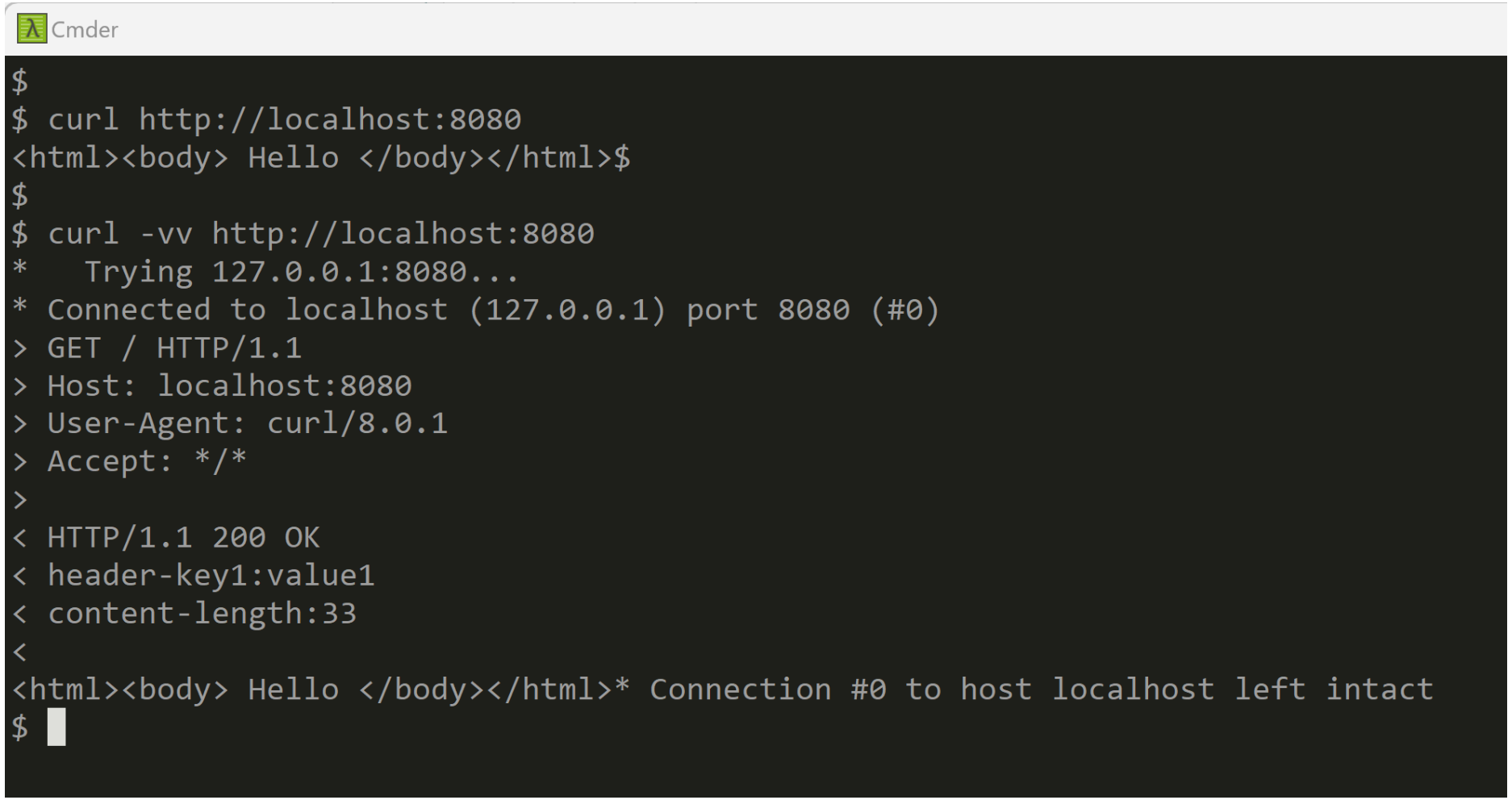


Debug Client & Server - Logs





Test using Curl (or Postman)



ANNEXE …
equivalent Server code using frameworks

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

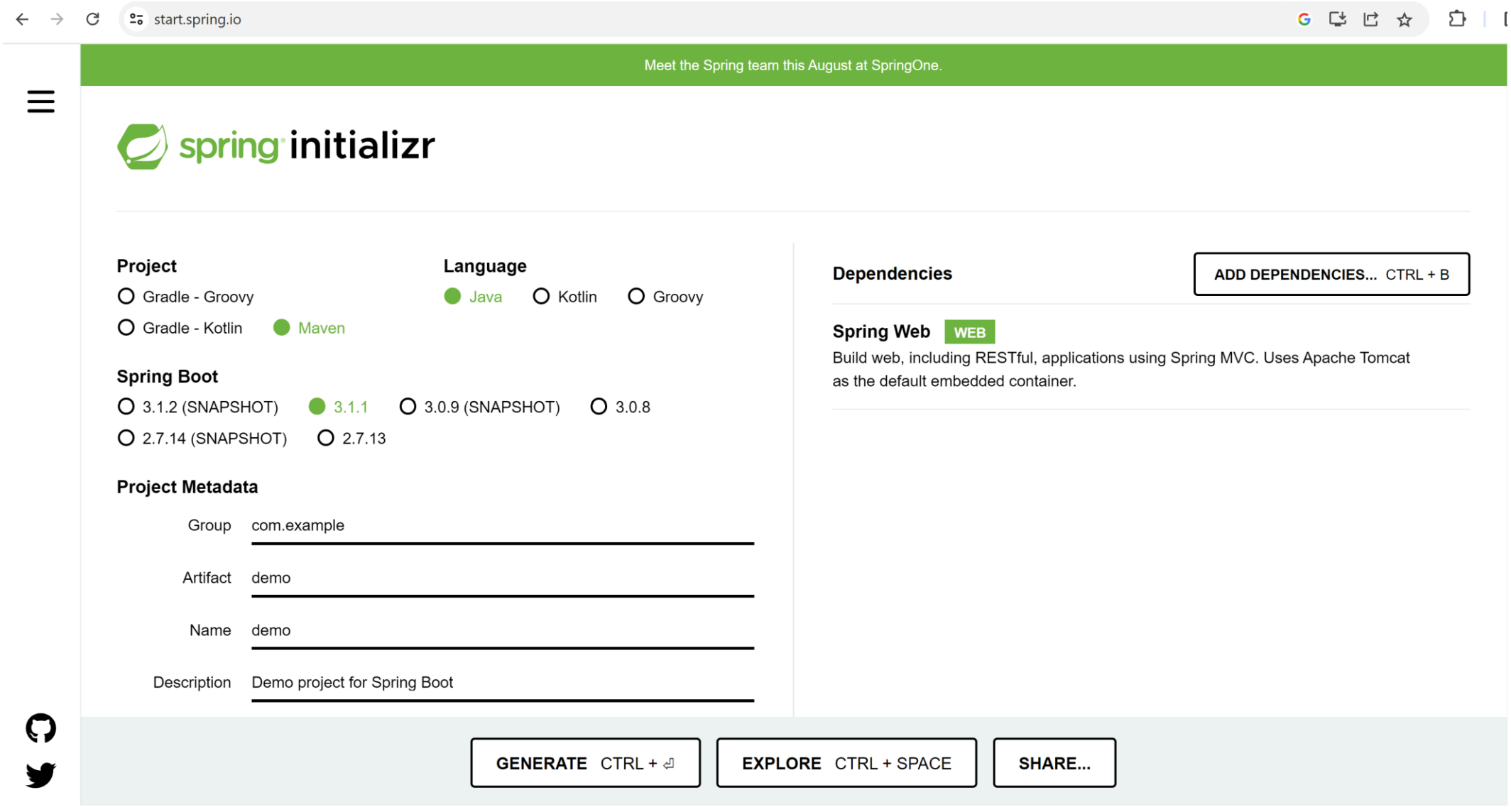
Actually more « <xml> </xml> » in maven pom.xml
than in java

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

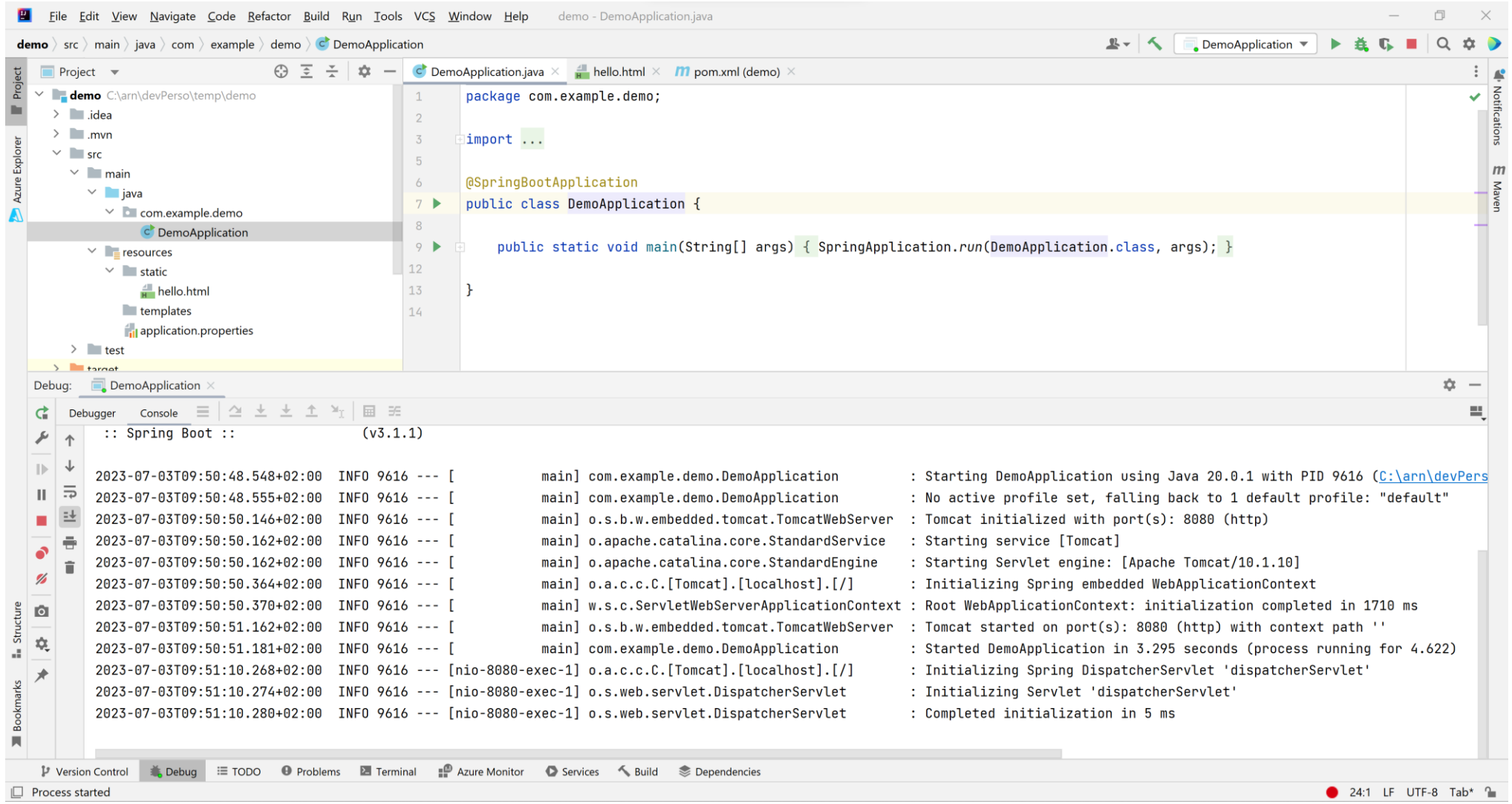
https://start.spring.io/

https://start.spring.io/

add dependency « WEB »



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



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 (unauthencated)
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, ..

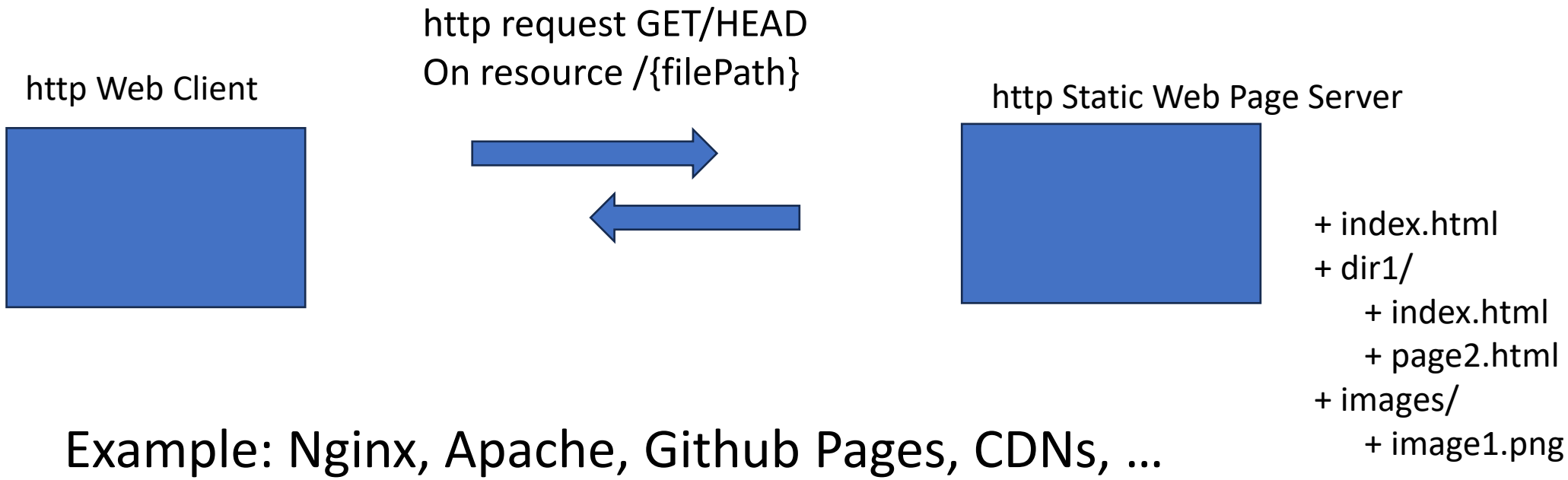
DELETE = delete a resource by its id/path
(idempotent : can retry)

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**



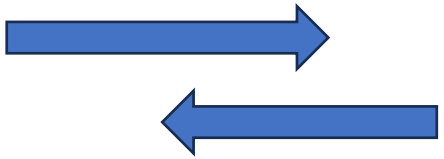
Example: Nginx, Apache, Github Pages, CDNs, …

Dynamic Page Server
(server-side generated web page)

Terminology: **Dynamic Page Server**

http request GET/POST/PUT/DELETE
On resource /{dynamicPage}/{id}?{query} {body}

Web Browser



http Dynamic Page Server

html page, generated on demand
Example: *.jsp, servlet, mvc view pages, …

Maybe also some static page.. (*.css, *.js,)

Museum of Dynamic Page Server Technologies

/htdocs / cgi-bin / *.sh

Personal
Home
Page

(not a decent langage)

***.SP**

Java Servlet & Jsp
(html in java / java in pseudo-html)

= 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() %>
```

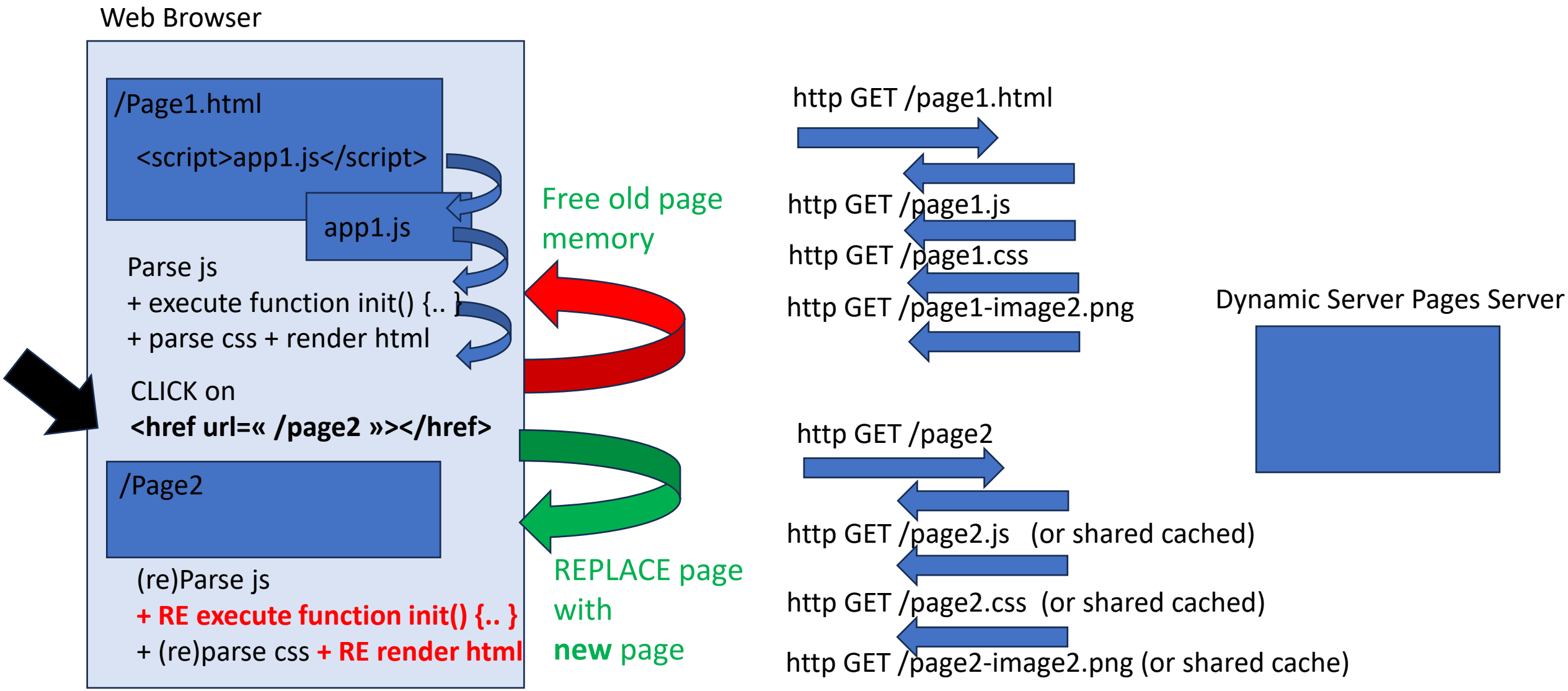
Choose your poison
Red pill / blue pill

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



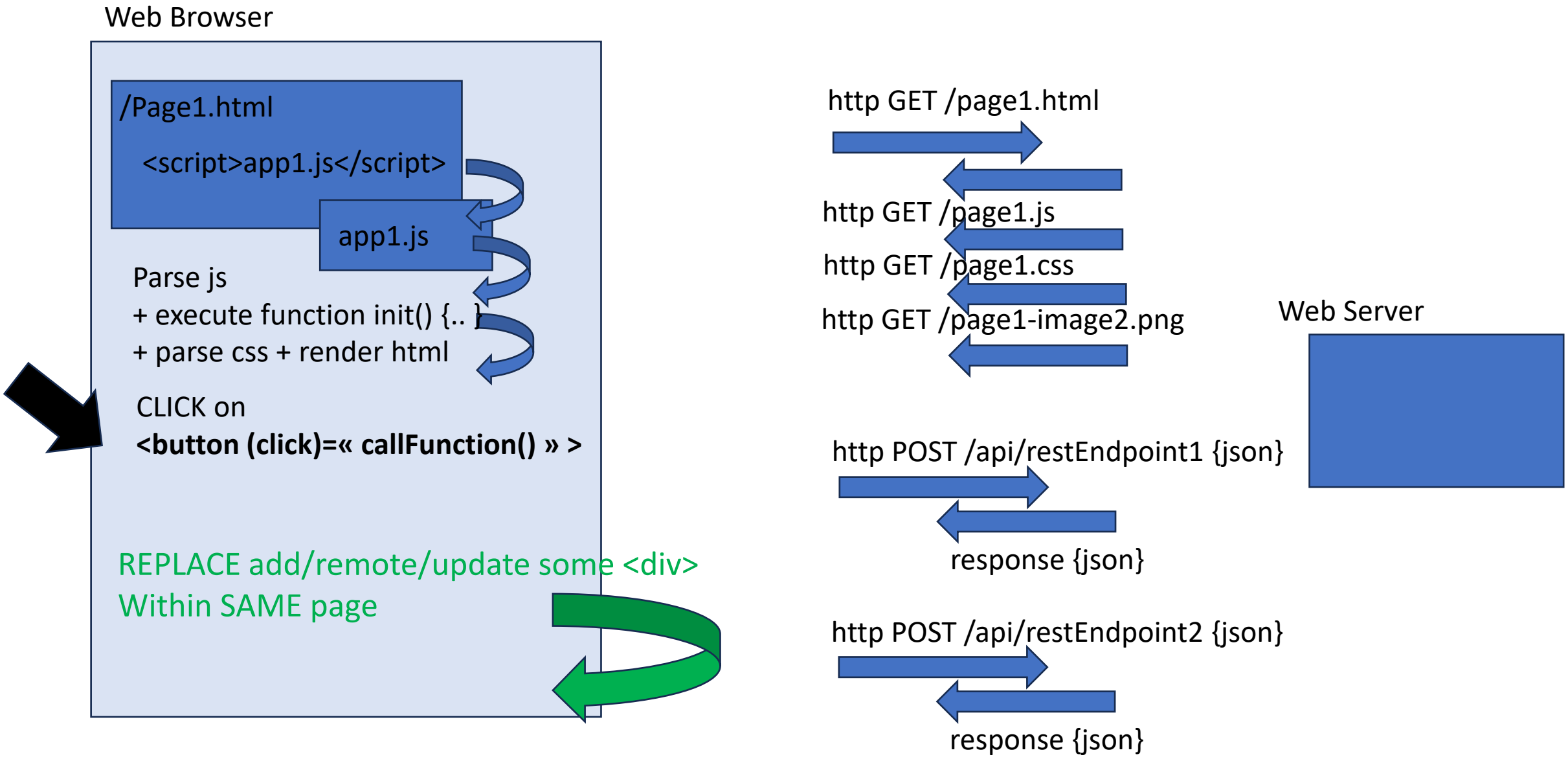
Solution to
Multiple Pages Application Problems ?

caching + optims + workarounds … ?

=>

SPA = Single Page Application !!

SPA = Single Page Application
… still dynamic, still Multiple Rest Json requests

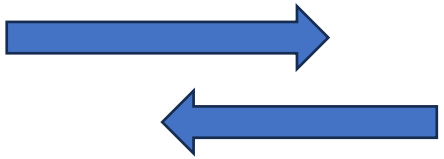


http API Server (cf Rest Api)

Terminology: **Http Api Server**

http request GET/POST/PUT/DELETE
On resource /{dynamicEntityPath}/{id}?{query}
Request body **{json}**

http Api Client



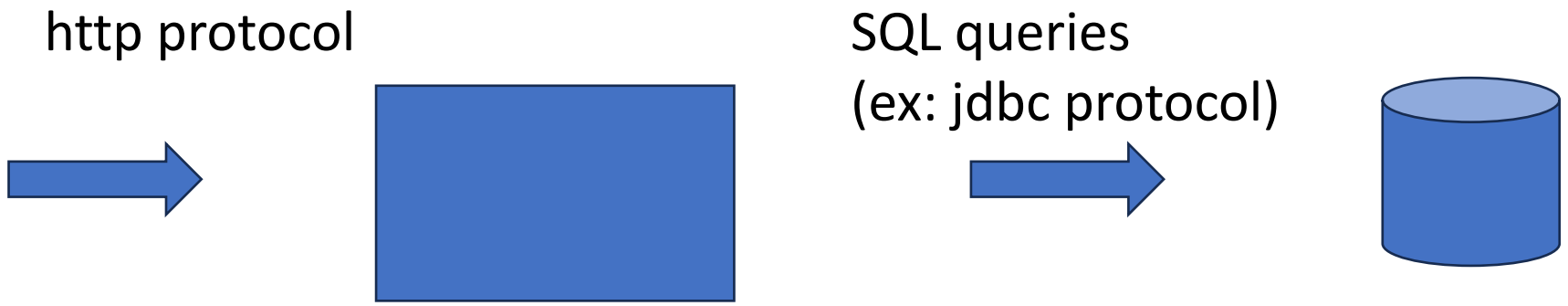
response body **{json}**

Http Api Server

Mostly *.json or *.xml !!!

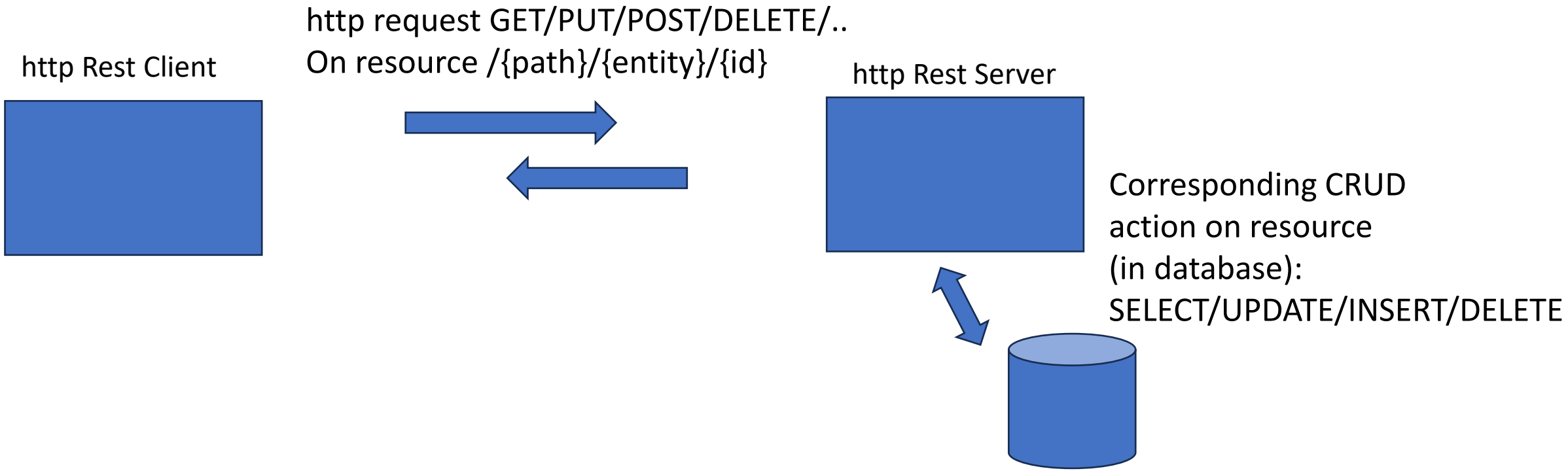
Maybe also few (static) pages *.html,*.js,*.css

http exposing a « CRUD » api
CRUD : Create – Read - Update - Delete

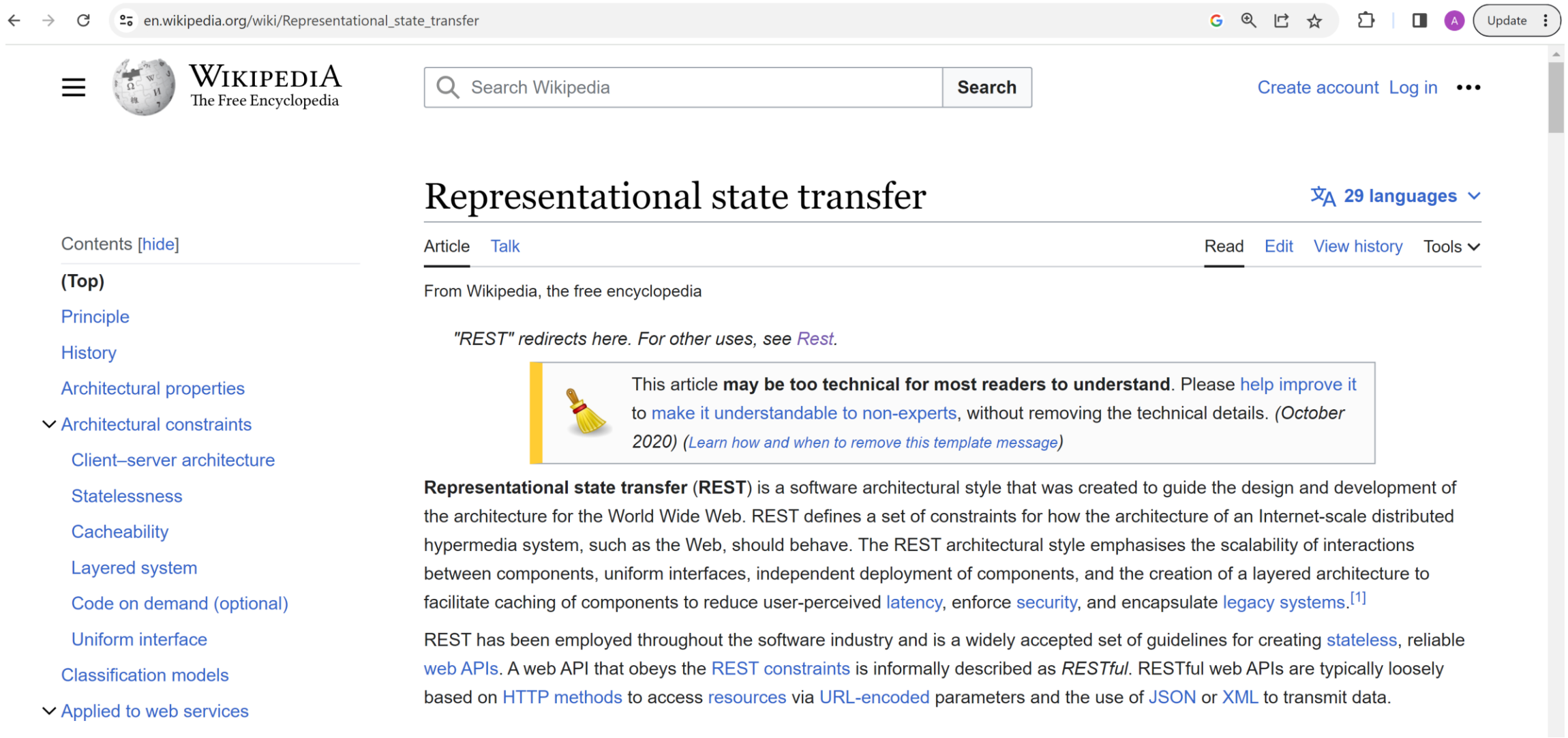


Create	http **POST** /{entityType} … body	**INSERT** into table T (..) values(..)
Read	http **GET** /{entityType}/{id}	**SELECT** * from T where id=?
	http **GET** /{entityType}?field=value	**SELECT** * from T where … limit 10
Update	http **PUT** /{entityType}/{id} .. body	**UPDATE** T set …=? where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

using Standard naming convention on Verb + Url
= follow « Rest » style
Rest = **Re**presentational **S**tate **T**ransfer



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



Rest-Json Api server in Springboot @Mapping ..

http endpoint		java rest controller class
http Request		java method
json request body		Java bean method parameter
json response body		Java bean method return

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

Java -> json

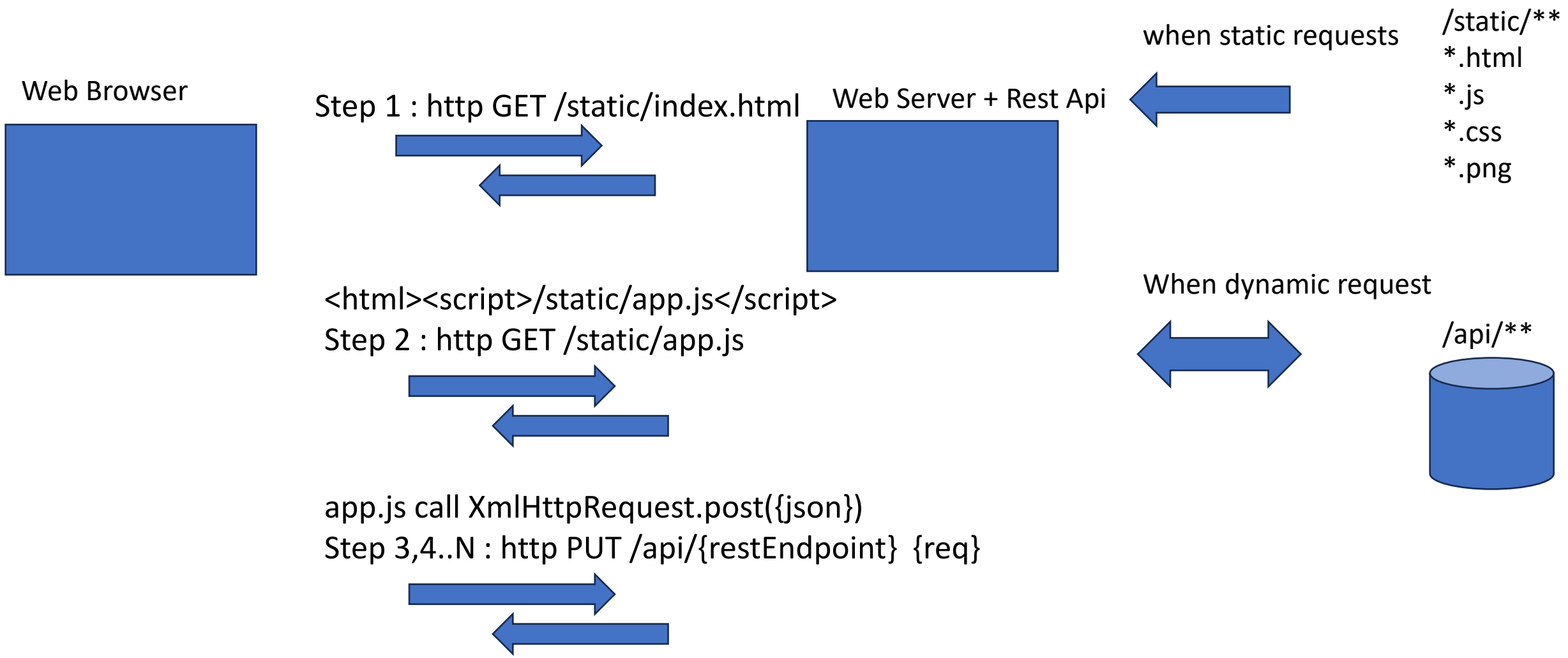
```
class ResponseDTO {
  public String respField;
}
```

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

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

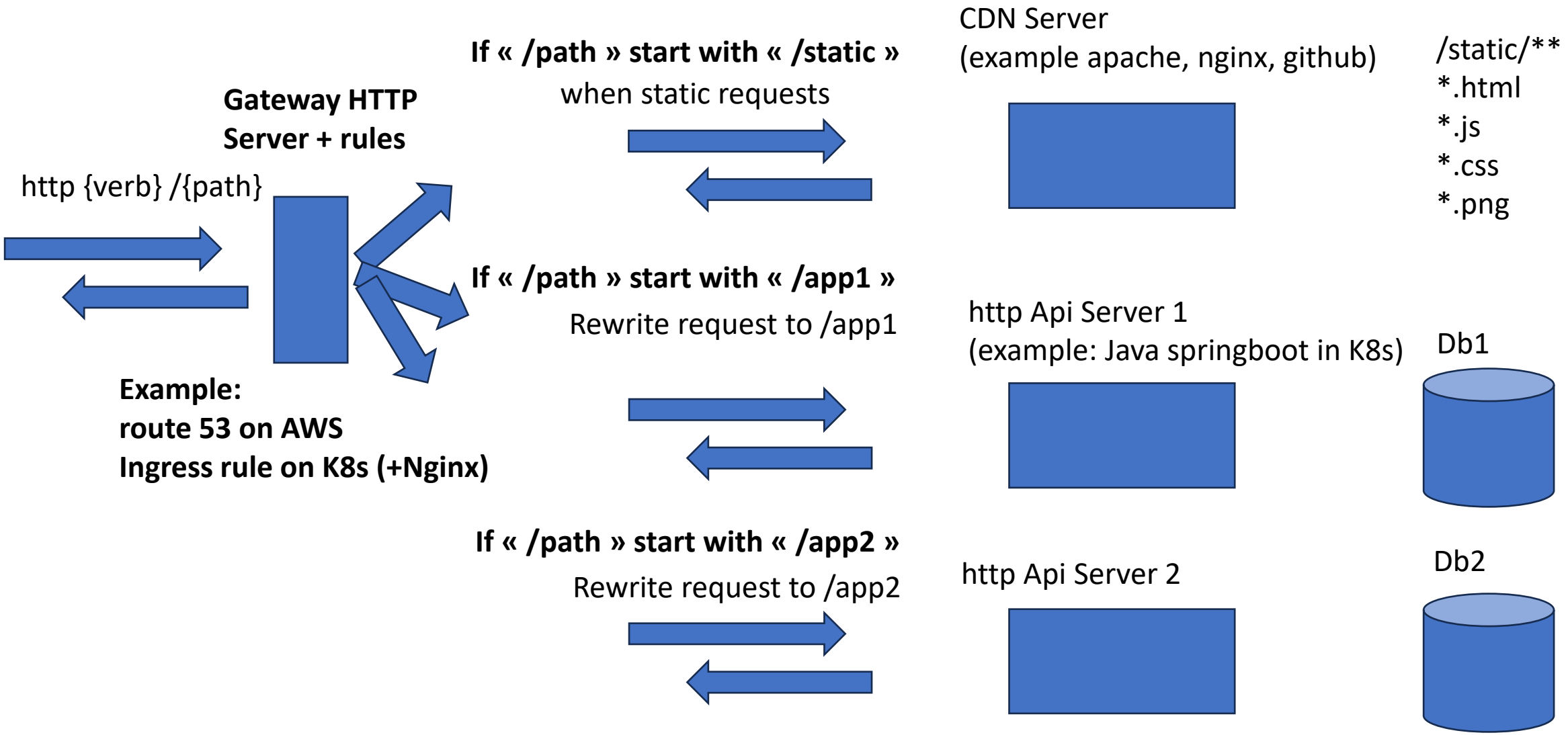
Example using NodeJs - Express

```
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 »

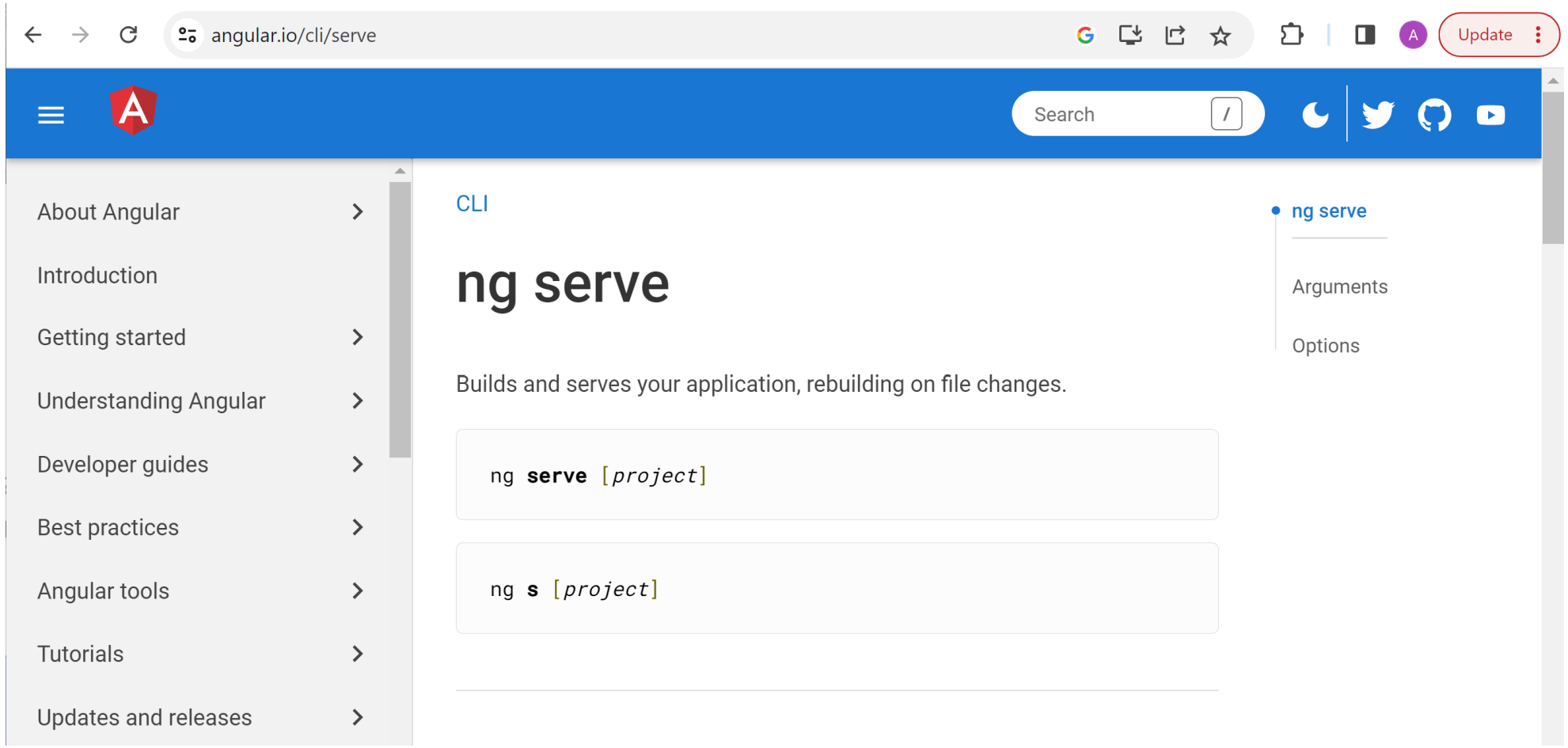
« Ng serve » has 1 , 2 or 3 roles !

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

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

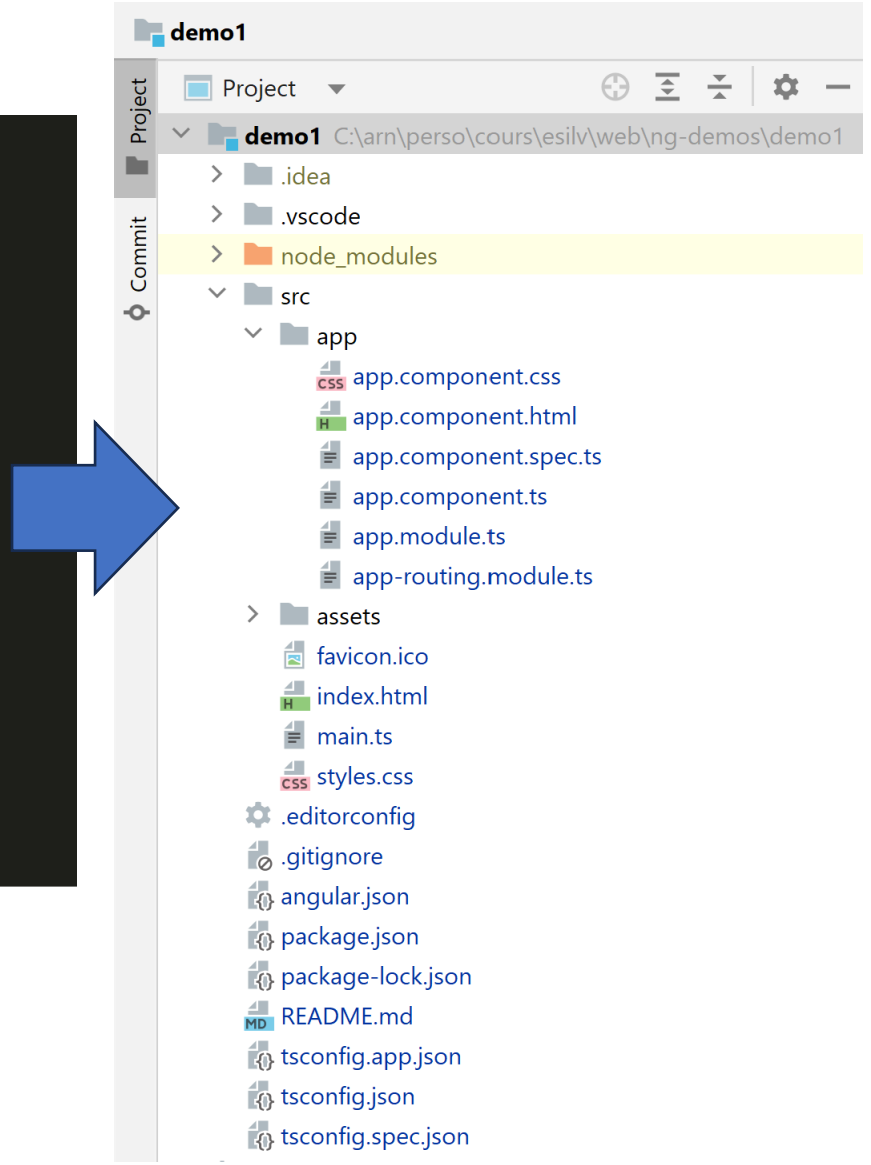
Role 3 : **proxy to backend** (eg. java)
requests in *.json /api/* redirected to backend api

https://angular.io/cli/serve



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

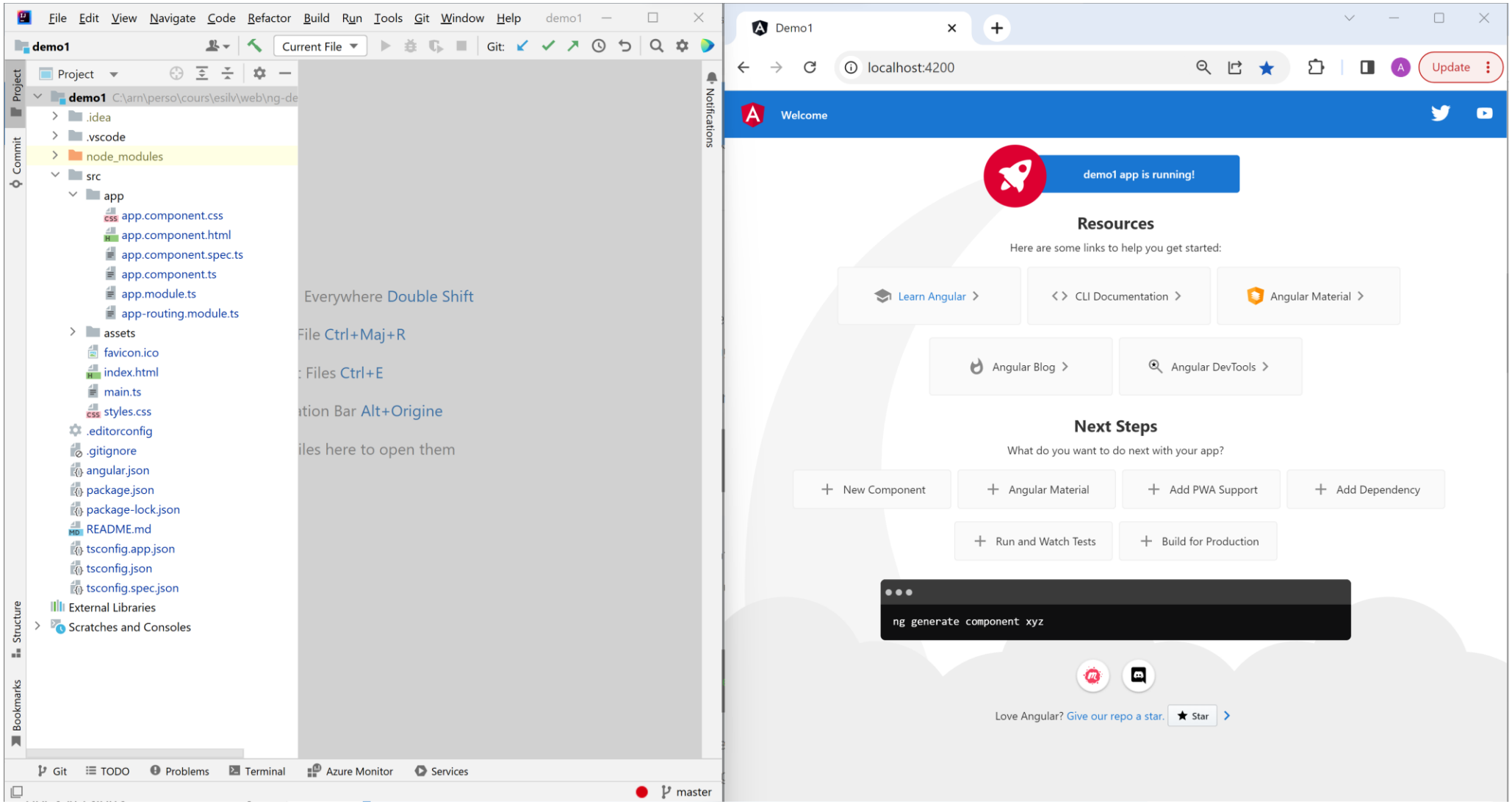
```
$ ng new
? What name would you like to use for the new workspace and initial project? demo1
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE demo1/angular.json (2695 bytes)
CREATE demo1/package.json (1036 bytes)
CREATE demo1/README.md (1059 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)
```



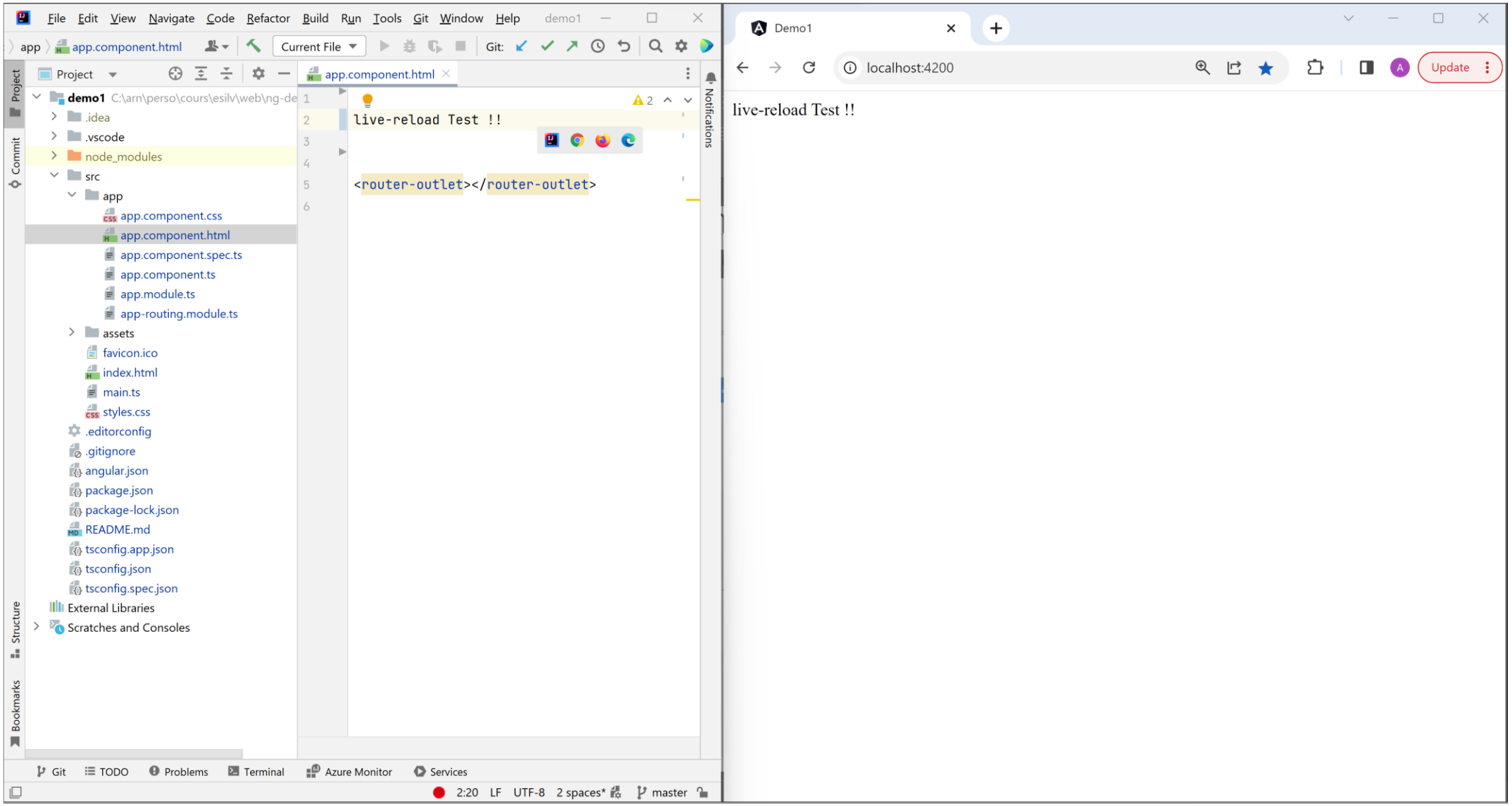
ng serve => http://localhost:4200
… live-reload on *.html, *.ts

```
$ 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.
```

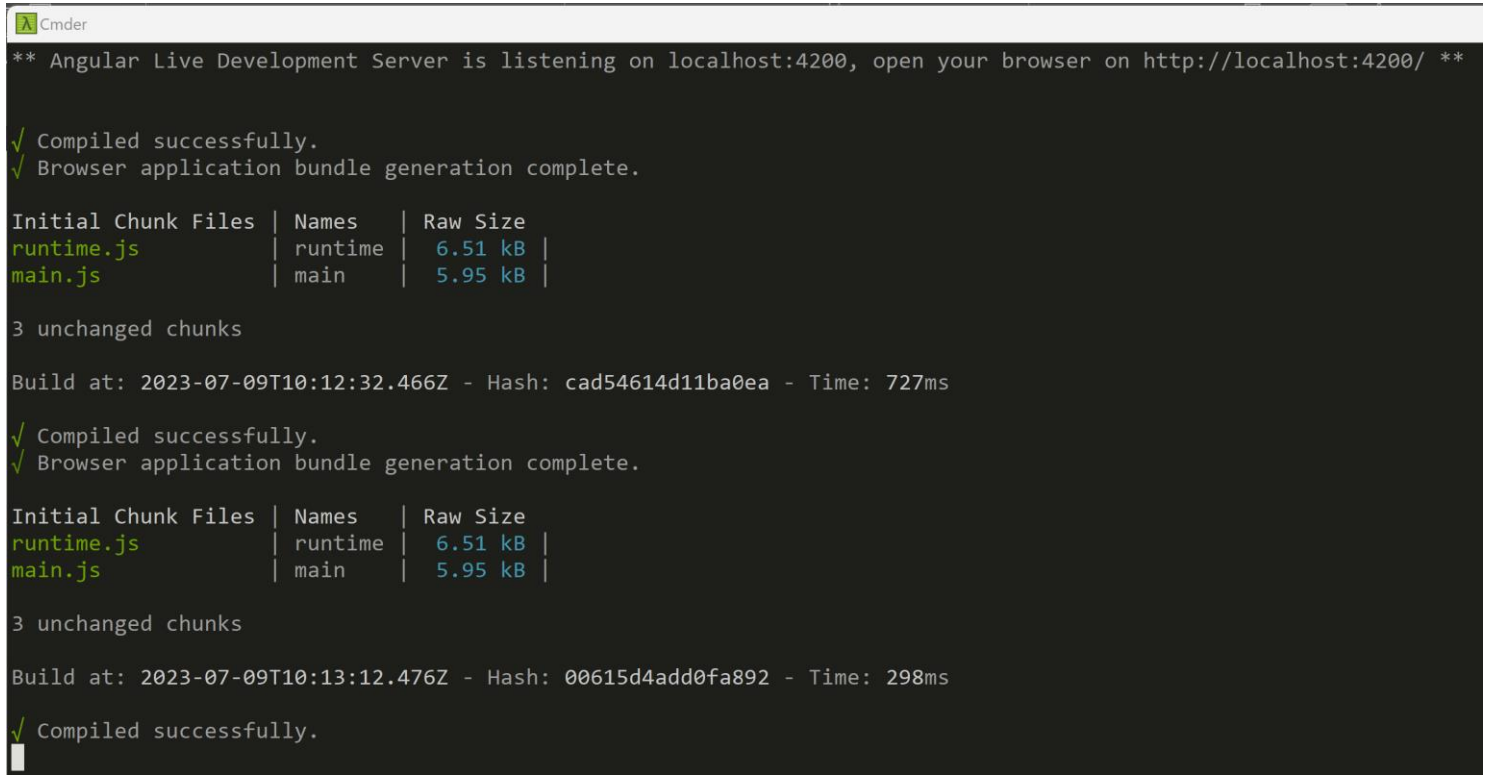
Demo
IntelliJ (left) + Chrome (right) + ng serve



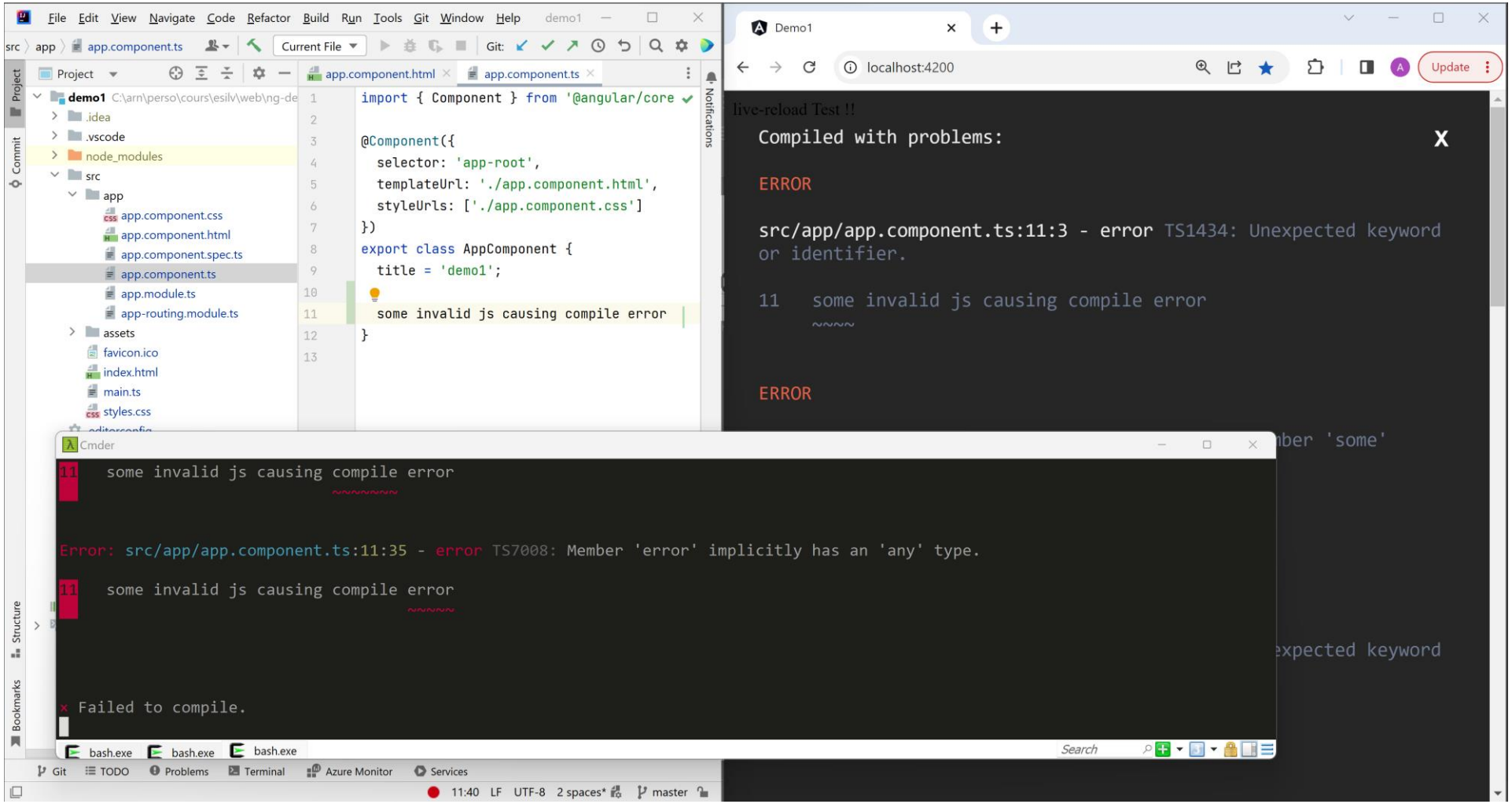
Demo … Live reload



Live Reload … --watch and compile !!

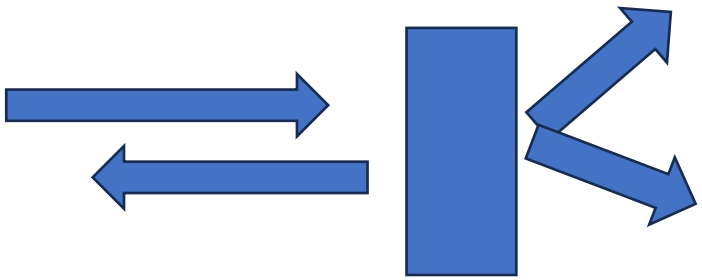


Demo live-reload … when compile error



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

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

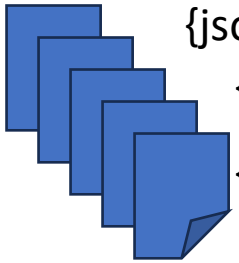


static *.html,*.js

app/server/
*.ts
static *.json response body

Real backend

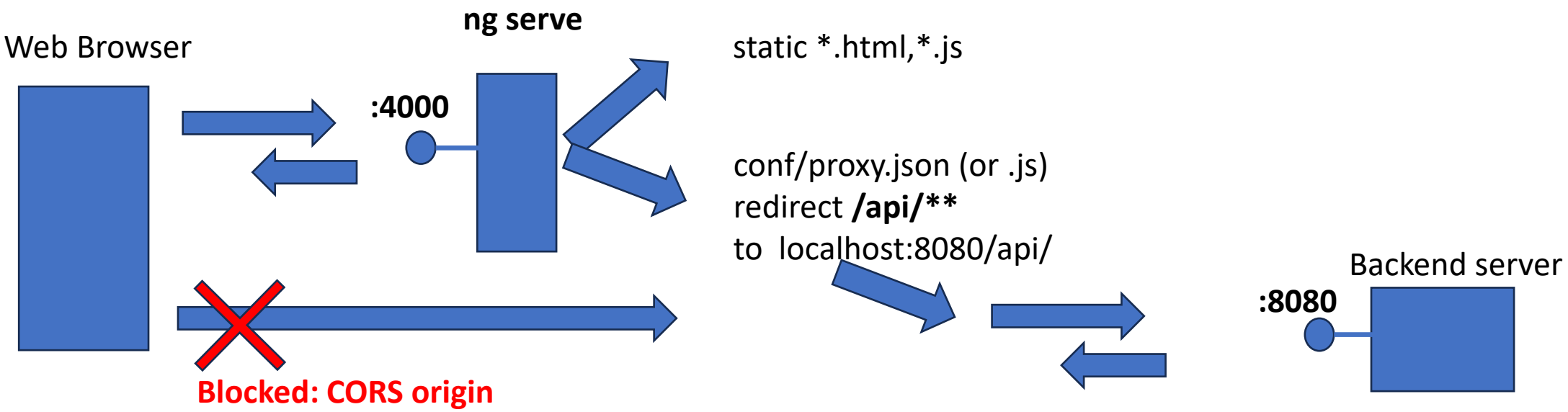
@Get(« /path/entity1 »
handler() {
NOT IMPLEMENTED YET
}



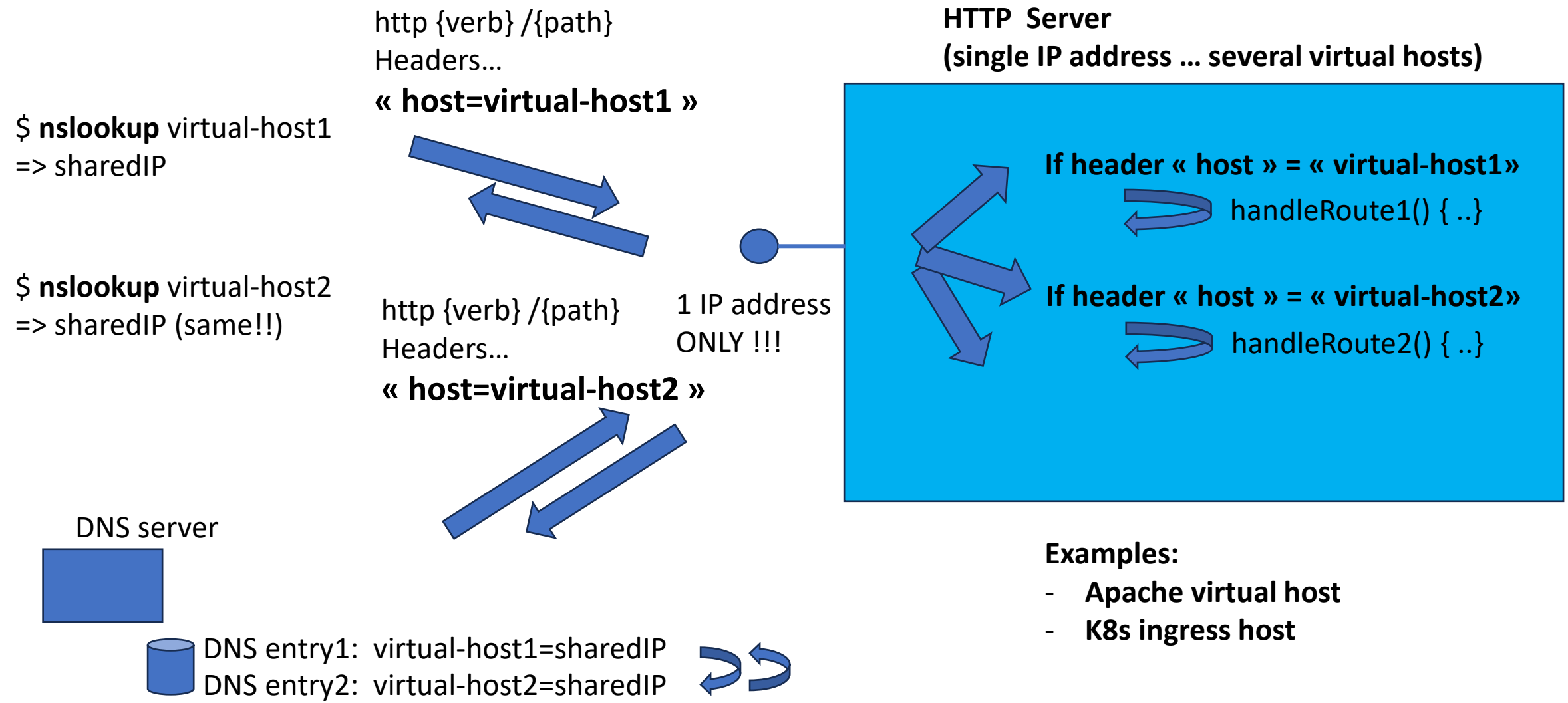
{json response for /path/entity1}
{json response for /path/entity2}
{json response for /path/query1}

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



Examples:

- **Apache virtual host**
- **K8s ingress host**

Proxy Server

Proxy server

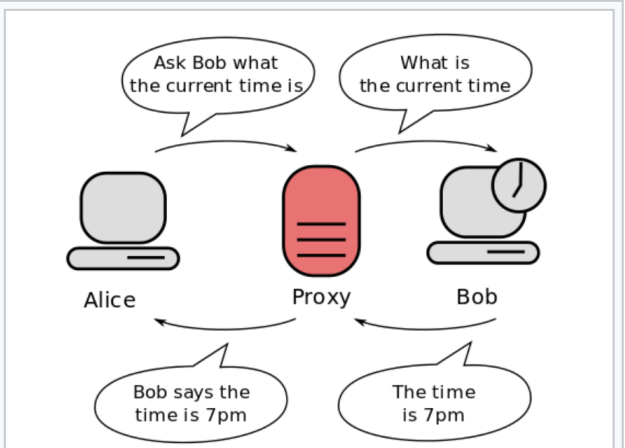
Contents [hide]

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.[1] 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.[2] 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.



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.

Types [edit]

A proxy server may reside on the user's **local computer**, or at any point between

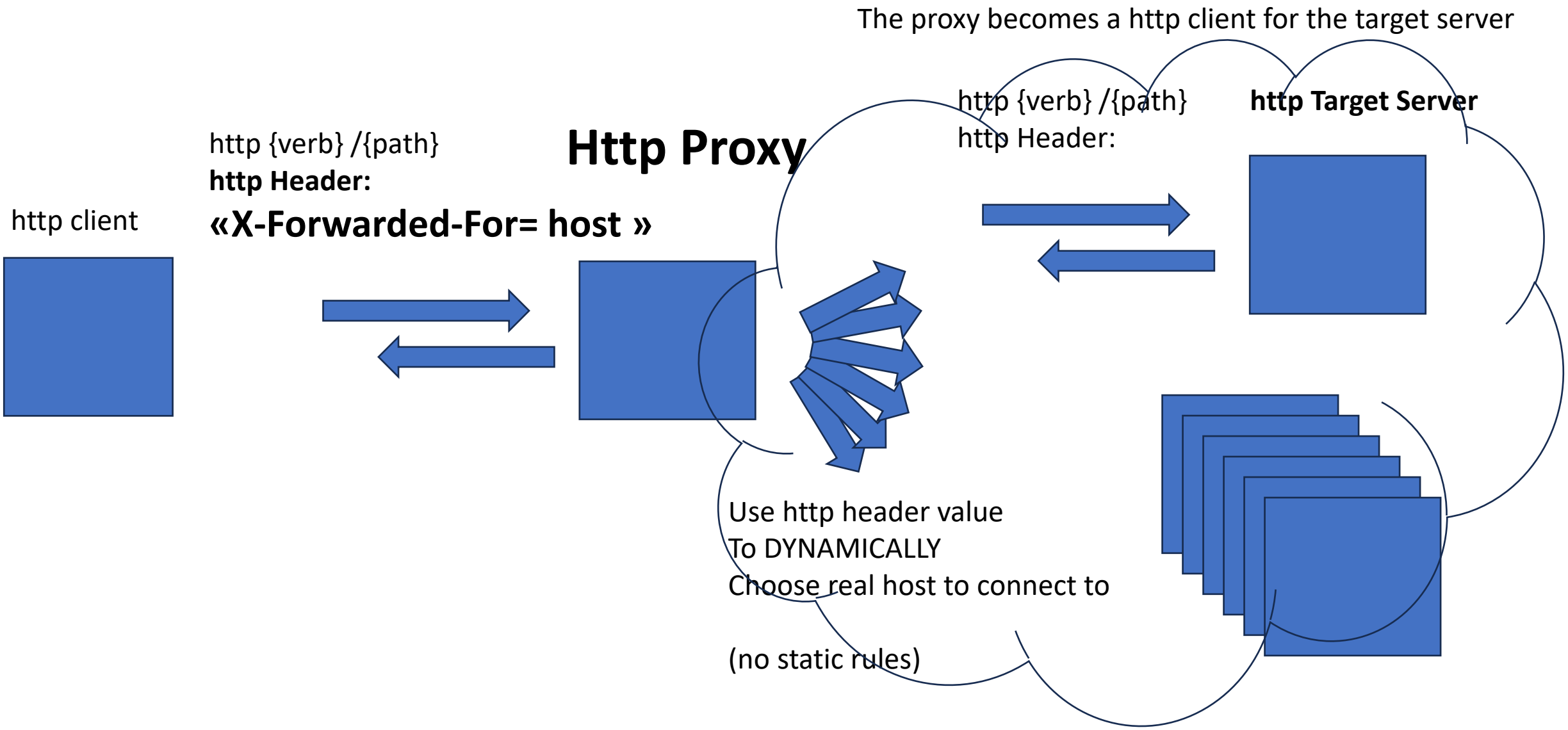
Forward Proxy ... Reverse Proxy ??

Some Terminology…

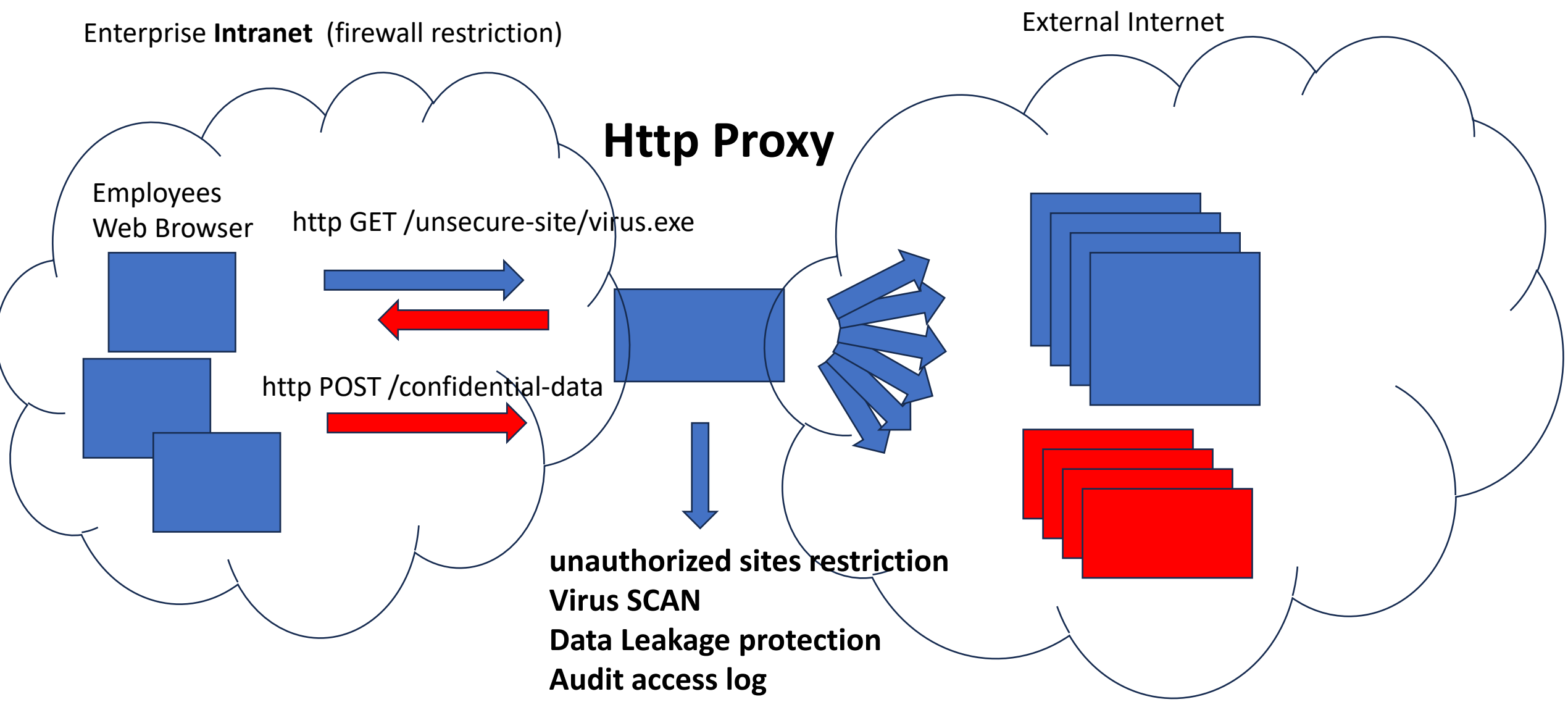
« **Reverse Proxy** » = **transparent** proxy = Gateway
looks like a standard server
(not seen as a http or socks proxy)
Example: Used for loadBalancing, ingress in K8s

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

Http Proxy


The proxy becomes a http client for the target server
http {verb} /{path}
http Header:
http target Server
http {verb} /{path}
http Header:
«X-Forwarded-For= host »
Http Proxy
http client
Use http header value
To DYNAMICALLY
Choose real host to connect to
(no static rules)

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»

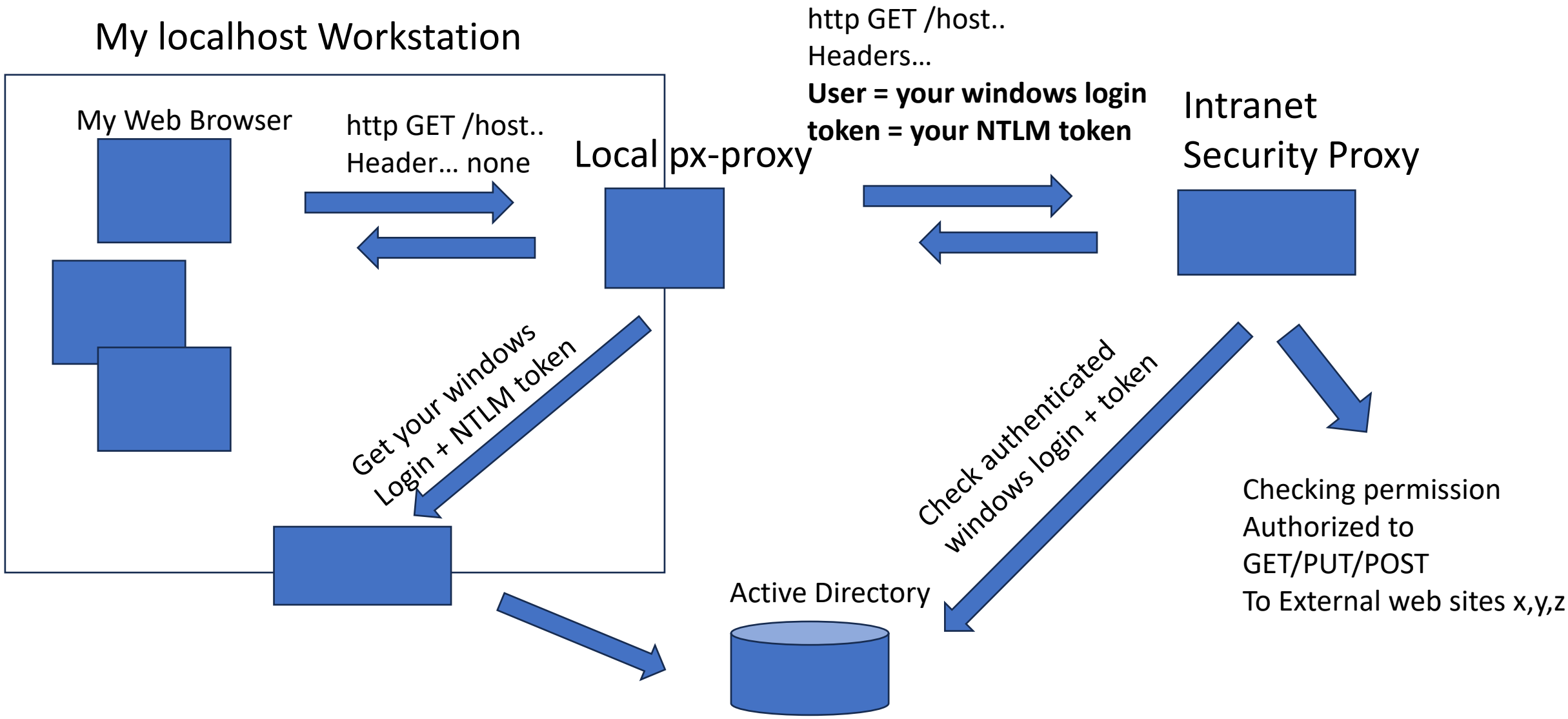
HTTP_PROXY is standard environment variable
There are also standard configuration file for many applications (example: <proxy>..</proxy> in maven settings.xml
And custom args per langage, example « java –Djava.net.proxy.http=proxyhost your-app.jar »

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


My localhost Workstation
My Web Browser
http GET /host..
Header… none
Local px-proxy
http GET /host..
Headers…
User = your windows login
token = your NTLM token
Intranet
Security Proxy
Get your windows
Login + NTLM token
Check authenticated
windows login + token
Checking permission
Authorized to
GET/PUT/POST
To External web sites x,y,z
Active Directory

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)

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

Runtime

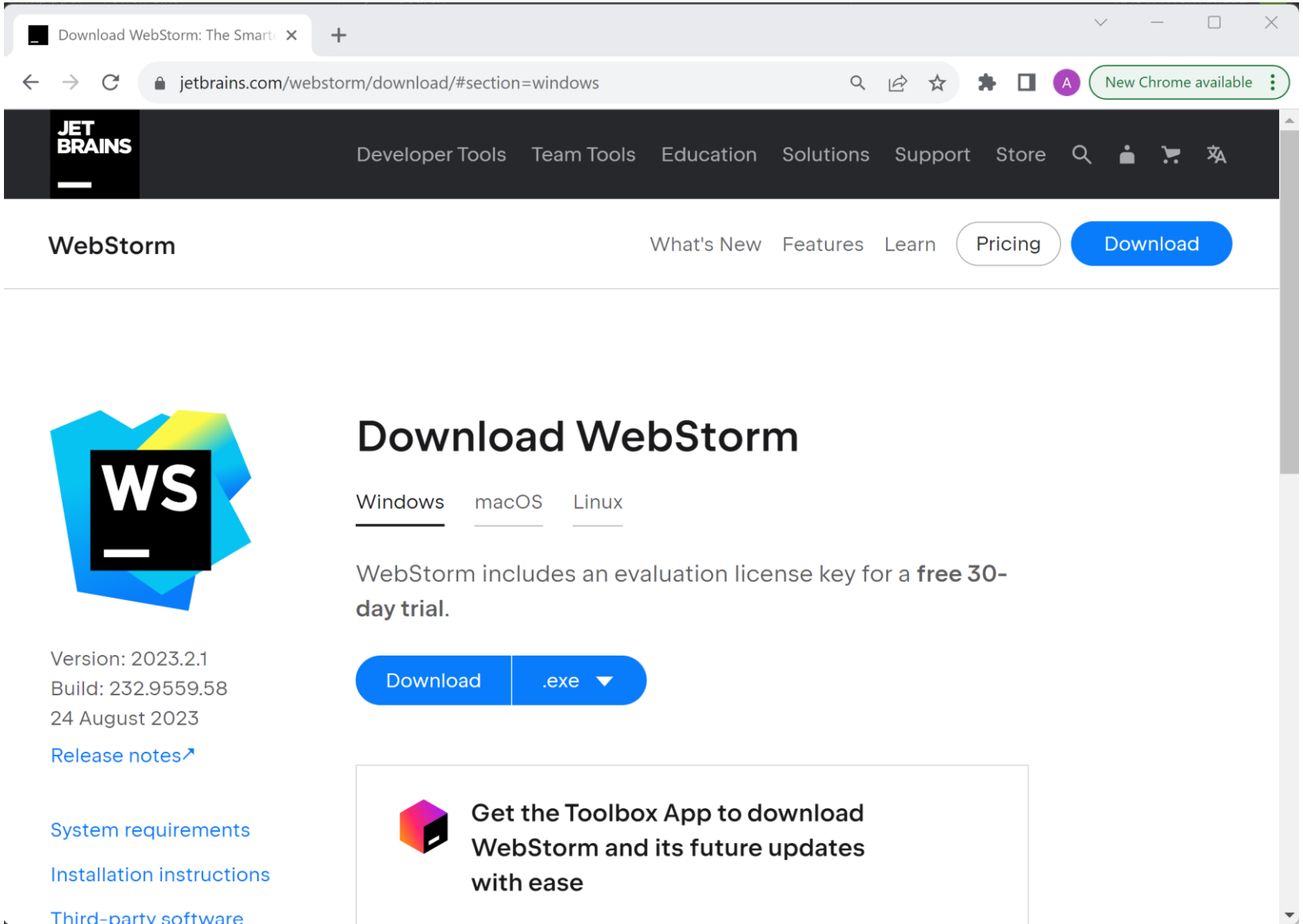
Nodejs
(+ npm)

Tool

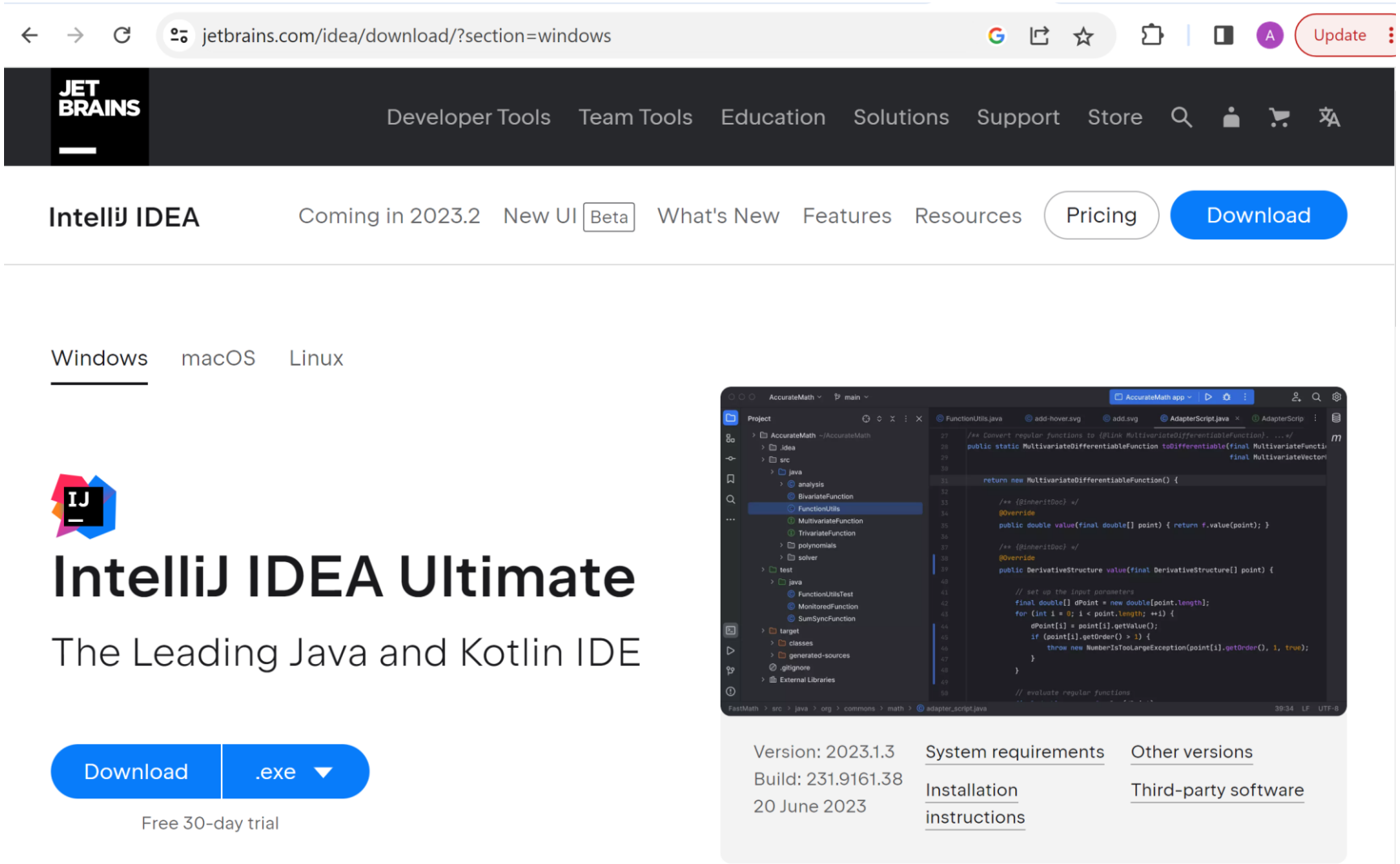
@angular-cli
(npm module in PATH)

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

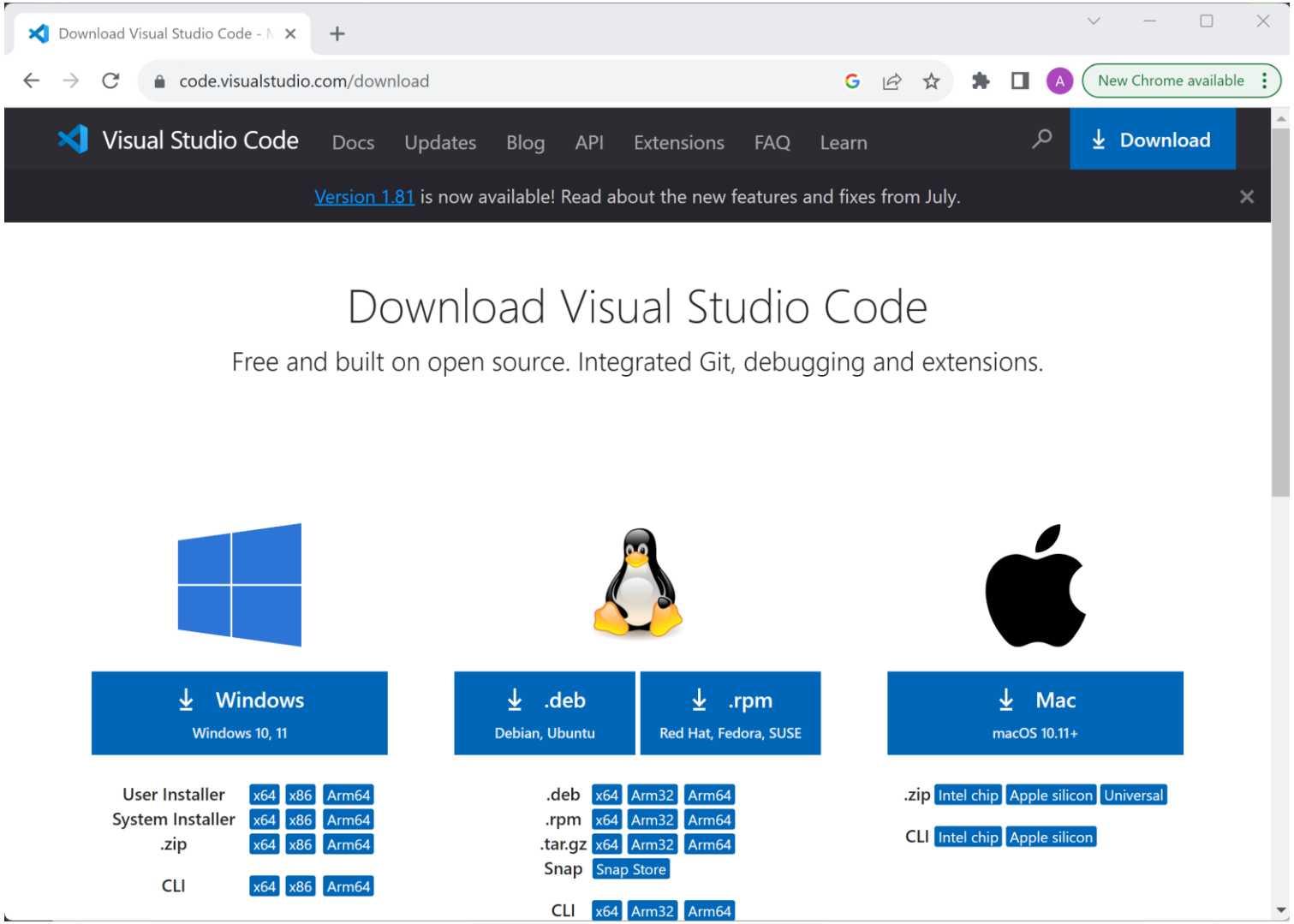
https://www.jetbrains.com/idea/download



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

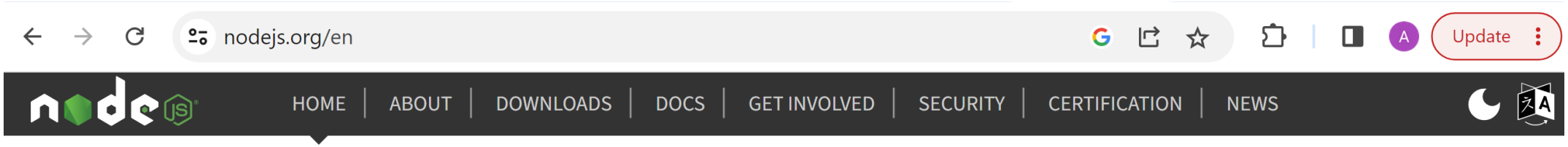


OR … Eclipse with *Web plugins
OR … VisualStudio Code



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

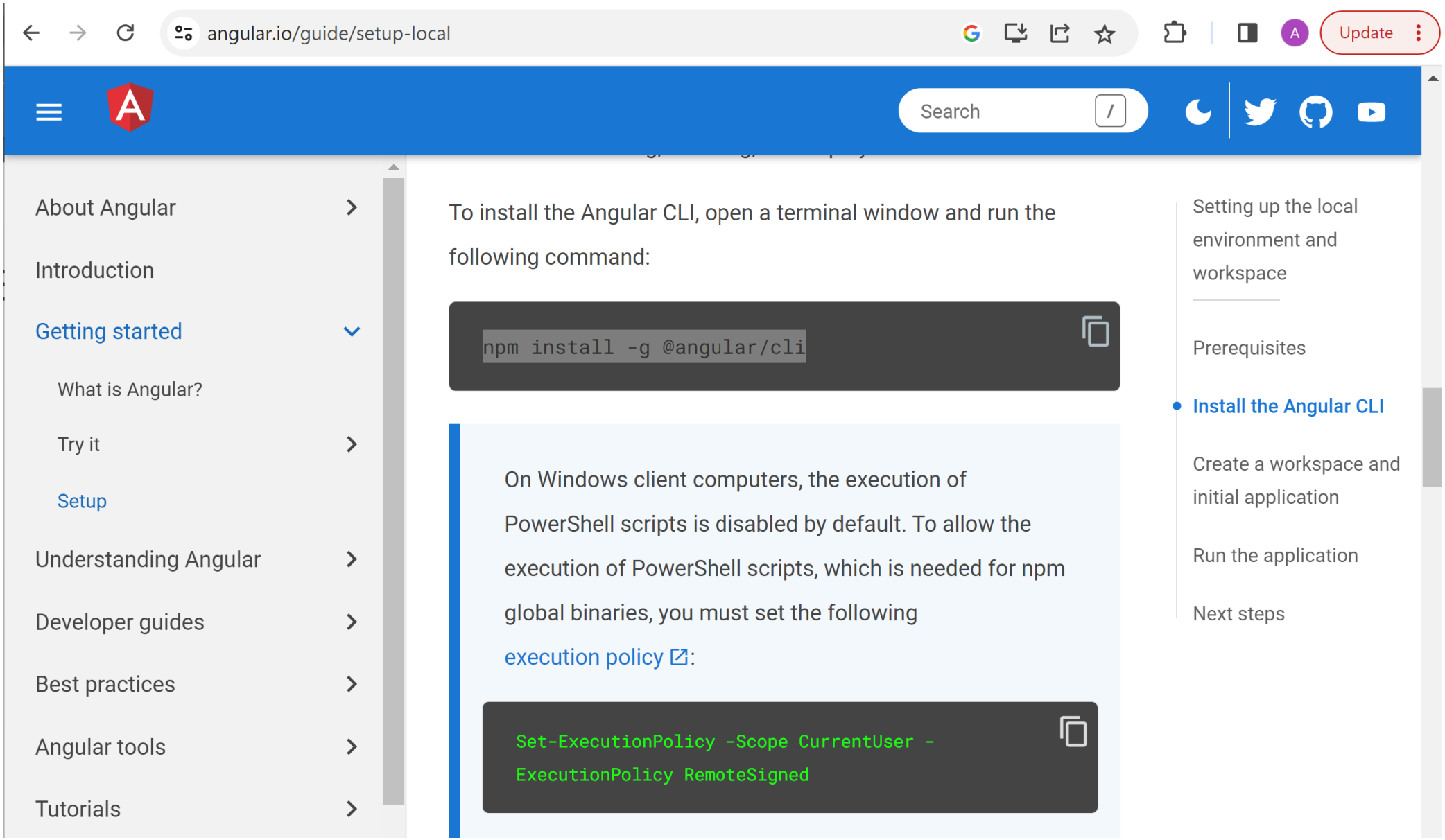
20.4.0 Current
Latest Features

Other Downloads | Changelog | API Docs

Other Downloads | Changelog | API Docs

For information about supported releases, see the release schedule.

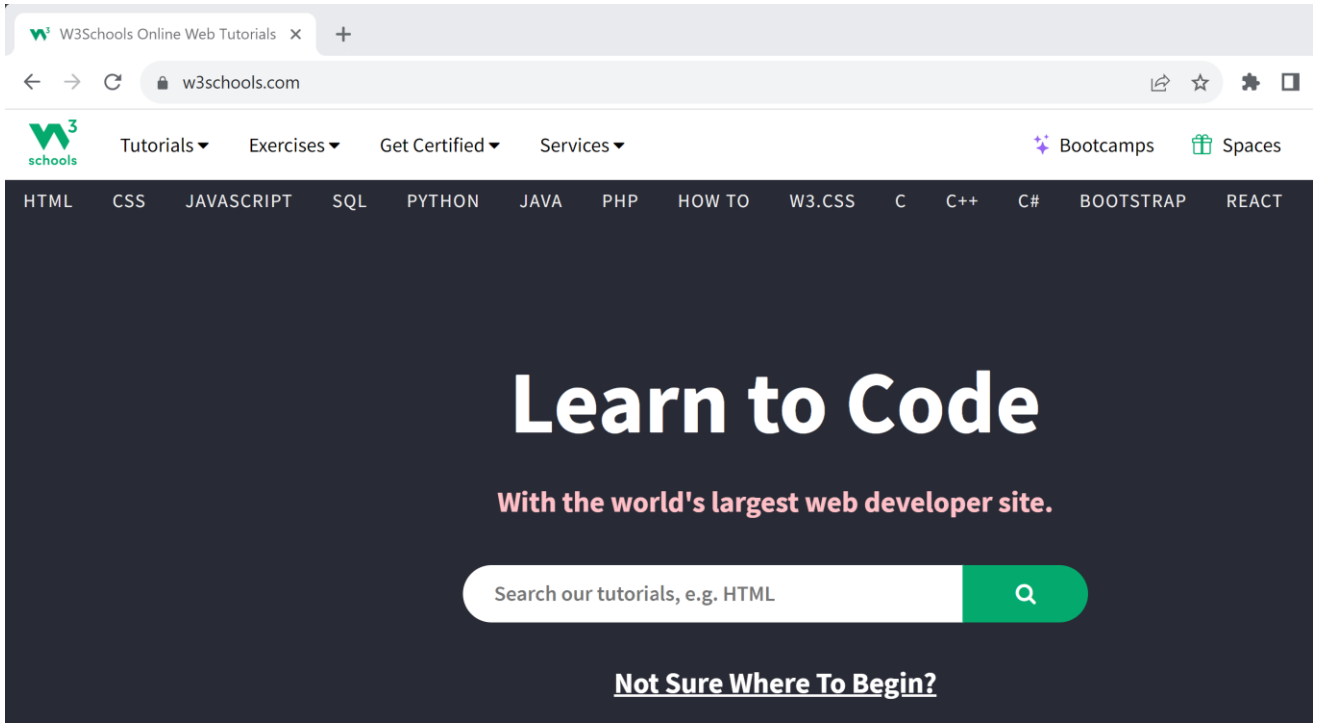
Pre-Requisite Install 3/3 : @angular/cli



Suggested Reading for Next Time

https://www.w3schools.com/

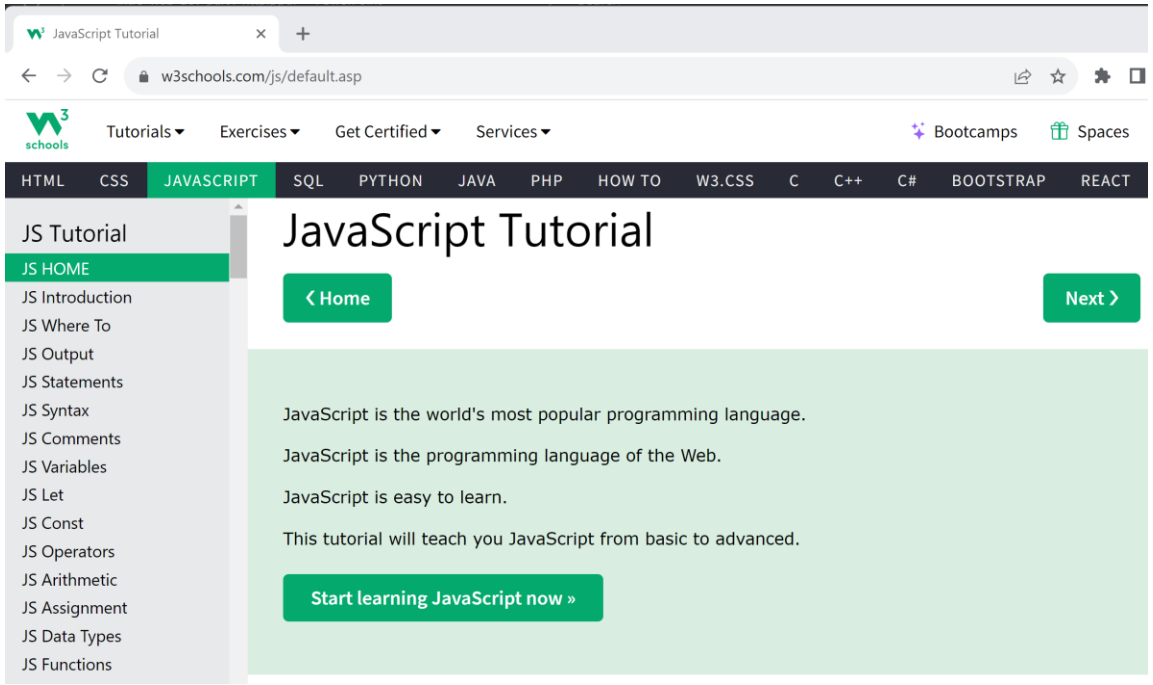
HTML + JavaScript + TypeScript + NodeJS + Angular …

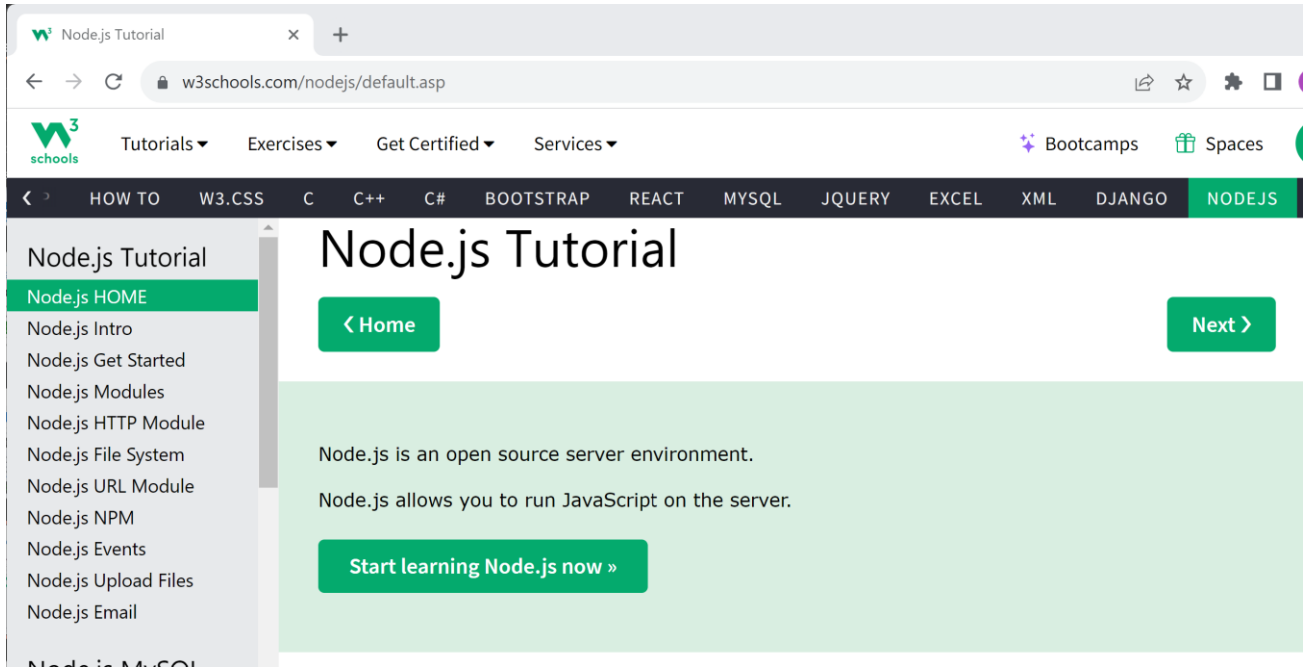


Suggested Reading for Next Time

https://www.w3schools.com/js

https://www.w3schools.com/nodejs





Questions ?

MindMap to Take Away

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

Links

arnaud.nauwynck@gmail.com

Course Esilv 2023

This document:

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