

Web Development Foundations

Web Programming – from HTTP to CGI Scripts

Assoc. Prof. Andrei PANU – profs.info.uaic.ro/andrei.panu/

* many thanks to Prof. Sabin-Corneliu Buraga for the content on these slides



Table of Contents

1. The interaction between Web client(s) and server(s)
2. How do we develop applications on the back-end



What is
the Web?

World Wide Web

an information space containing elements of interest, called **resources**, denoted by global identifiers – **URI/IRI**

details at www.w3.org/TR/webarch/
W3C Recommendation (2004)

Web Resources

Aspects of interest:

identification

interaction

representation by using data formats

Web Resources

Aspects of interest:

identification

URI/IRI

protocol: HTTP

interaction

representation by using data formats

markup language(s)




How about the interaction
between Web client(s) and server(s)?

HTTP

HyperText Transfer Protocol

based on the TCP/IP stack



more details in
a future lecture

HTTP

situated on the application layer

hypertext/hypermedia transfer
(HTTP – HyperText Transfer Protocol)

reliable transport via sockets
(TCP – Transmission Control Protocol)

network interconnection + data routing
(IP – Internet Protocol)

access control to the data transmission
medium (MAC – Medium Access Control)

HTTP

HyperText Transfer Protocol

a reliable request/response protocol

standard access ports: 80 and 443
(the latter for HTTPS)

HTTP

HTTP/1.1

Internet standard: RFC 2616 (1999)

from 2022, defined by RFC 9110—9112

[httpwg.org/specs/
devdocs.io/http/](http://httpwg.org/specs/devdocs.io/http/)

tutorial: http.dev

HTTP

HTTP/2.0

RFC 9113 (2022)

focused on performance:

binary messages, TCP connection reuse (a single connection per host), multiplexing (many parallel streams), sending messages to the client (server push), header compression (HPACK)

<http2.github.io>

<http2-explained.haxx.se>

www.tunetheweb.com/blog/http-versus-https-versus-http2/

HTTP

HTTP/3.0

RFC 9114 (2022)

HTTP over QUIC (Quick UDP Internet Connections)

UDP, data stream multiplexing, data encryption, quick connection establishment

explanations:

http.dev/3

www.smashingmagazine.com/2021/08/http3-core-concepts-part1/
http3-explained.haxx.se

HTTP - Architecture

Web Server

daemon – “protective spirit”

Web Client

browser, robot (Web crawler), multimedia player, household appliance, conversational assistant, native application (desktop, mobile)...



Web Servers:

Apache HTTP Server, NGINX, Eclipse Jetty, Gunicorn, Lighttpd, Microsoft Internet Information Services, etc.

HTTP - Architecture

Web Client

Web browsers: Mosaic->Netscape->Mozilla.org->Firefox,
MS Internet Explorer, Chromium, Safari, Opera GX, etc.

tools: curl, wget,...

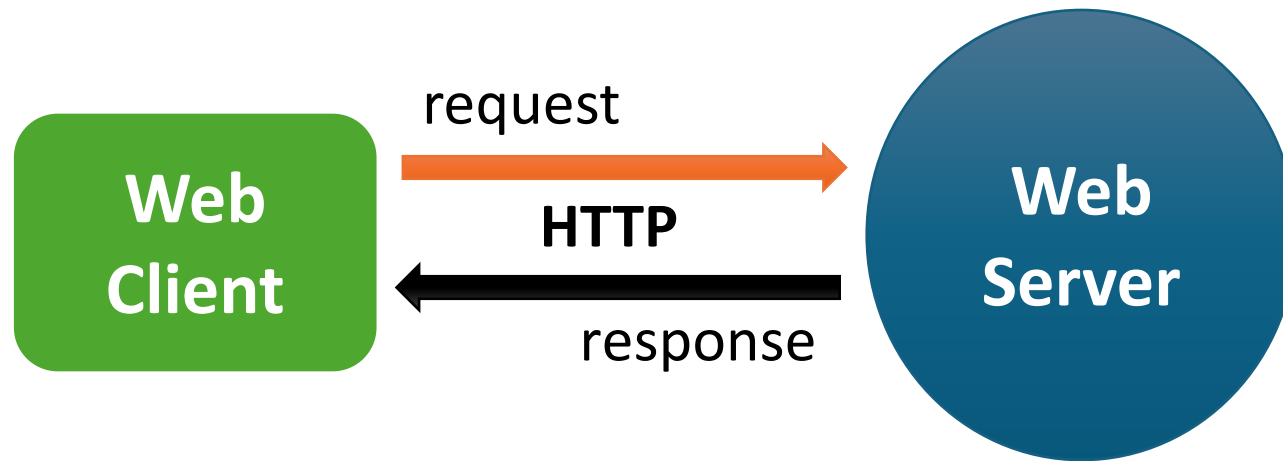
search engines robots: Slurp (Yahoo!), Googlebot

native applications offering various functionalities implemented by
service providers: e-commerce, finance, tourism, entertainment,
education, etc.

HTTP

request and response

accessing – possibly, changing – a resource
representation by using its URI



HTTP – Concepts

Message

base unit of the HTTP communication
(request or response)

HTTP – Concepts

advanced

Intermediary

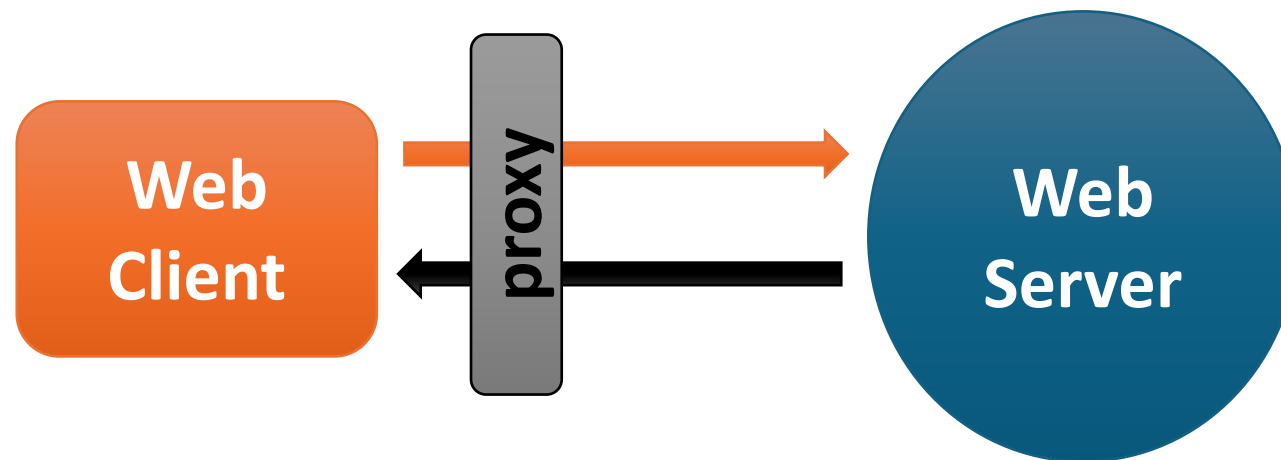
proxy
gateway
tunnel

HTTP – Concepts

advanced

Proxy

located in the client/server proximity
having the role of both server and client



HTTP – Concepts

advanced

Proxy

forward proxy

intermediary for a group of nearby clients
acts on behalf of clients

reverse proxy

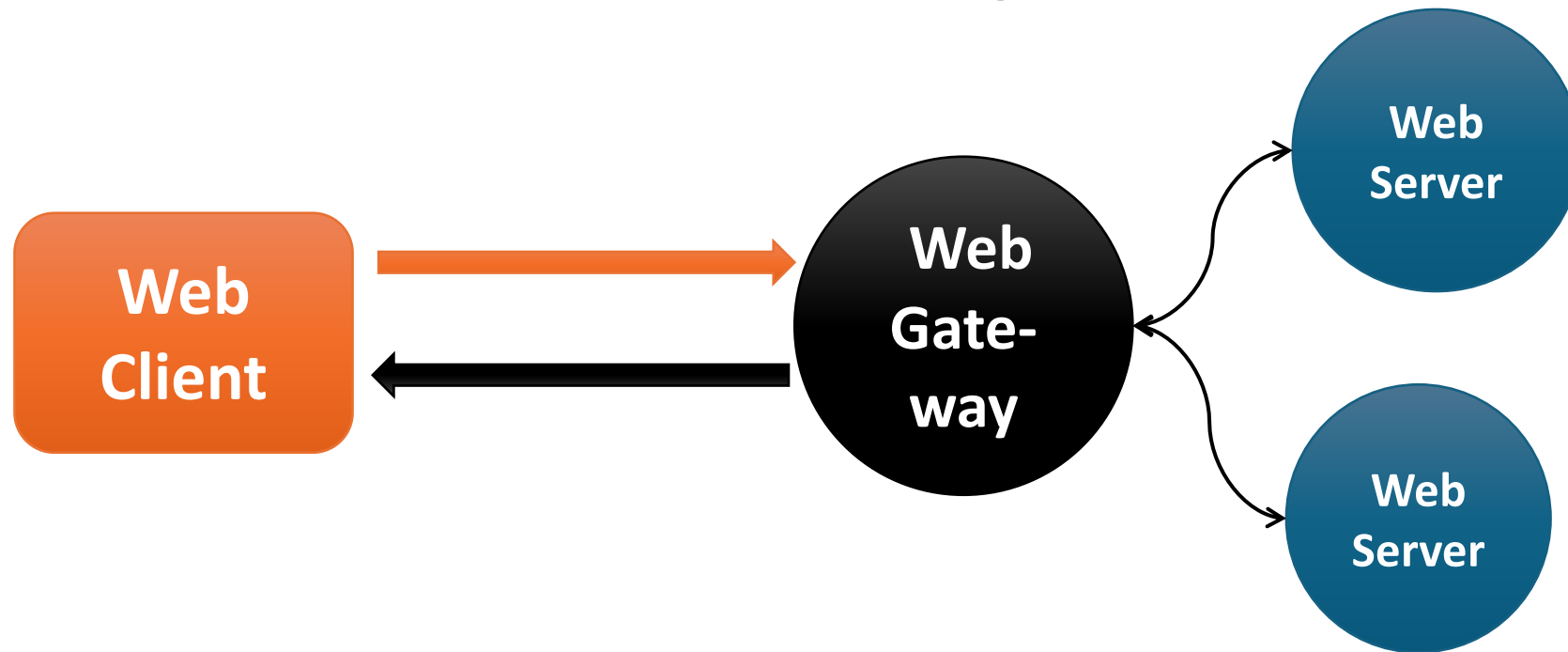
intermediary for a group of nearby servers

HTTP – Concepts

advanced

Gateway

intermediary hiding the target (origin) server
the client has no knowledge about this



HTTP – Concepts

advanced

Gateway

can assure:

- traffic distribution across servers – load balancing
- short-term data storage – caching
- message or request translation (e.g., HTTPS->HTTP)
- other negotiation operations – role of mediator/broker

HTTP – Concepts

advanced

Tunnel

retransmits – usually, encrypted – HTTP messages

HTTP – Concepts

advanced

Tunnel

context:

HTTPS protocol – ensures “secure” communication
HTTP via TLS (Transport Layer Security)

authentication based on digital certificates
+ bidirectional data encryption

visual tutorial: howhttps.works

Details about an HTTPS connection offered by a browser's developer tools

▼ Connection:

Protocol version: "TLSv1.3"

Cipher suite: "TLS_AES_128_GCM_SHA256"

Key Exchange Group: "x25519"

Signature Scheme: "ECDSA-P256-SHA256"

used encryption

▼ Host validator.w3.org:

HTTP Strict Transport Security: "Disabled"

Public Key Pinning: "Disabled"

▼ Certificate:

▼ Issued To

Common Name (CN): "sni.cloudflaressl.com"

Organisation (O): "Cloudflare, Inc."

Organisational Unit (OU): "<Not Available>"

information about
the digital certificate

▼ Issued By

Common Name (CN): "Cloudflare Inc ECC CA-3"

Organisation (O): "Cloudflare, Inc."

Organisational Unit (OU): "<Not Available>"

▼ Period of Validity

Begins On: "Mon, 06 Feb 2023 00:00:00 GMT"

Expires On: "Tue, 06 Feb 2024 23:59:59 GMT"

▼ Fingerprints

SHA-256 Fingerprint: "33:0C:6D:2C:73:DF:45:24:BE"

SHA1 Fingerprint: "6C:C9:8A:A3:48:99:C3:50:AD:8C"

advanced

HTTP – Concepts

Cache

local storage area – in memory, on disc –
for the messages (data)

server-side and/or client-side

HTTP – Concepts

Cache

local storage area – in memory, on disc –
for the messages (data)

future requests for that data can be served faster

context: ensuring Web applications' performance

HTTP – Messages

HTTP message = **header + body**

HTTP – Messages

Header

includes a set of fields

field-name ":" [**field-value**] CRLF

CR = *Carriage Return* \r – code 13

LF = *Line Feed* \n – code 10

HTTP – Messages

HTTP request

includes a set of fields

Method **Request-URI** **ProtocolVersion** **CRLF**
[**Message-header**] [**CRLF MIME-data**]

```
GET /web-development-foundations/ HTTP/1.1 CRLF
Host: edu.info.uaic.ro
```

HTTP – Messages

HTTP response

HTTP-version **DigitDigitDigit** **Reason** **CRLF**
[**Message-header**] [**CRLF MIME-data**]

HTTP/1.1 200 OK CRLF ...

HTTP – Methods

GET

request – performed by a client – to access
a resource's representation

HTTP – Methods

GET

request – performed by a client – to access
a resource's representation

HTML document, CSS stylesheet,
image in JPEG/PNG format, vector illustration as SVG,
JavaScript program, data in JSON (JavaScript Object Notation) format,
3D model, RSS (XML) news feed, PDF presentation, ZIP archive, video, ...

HTTP – Methods

HEAD

similar to GET
usually, offers only meta-data

HTTP – Methods

HEAD

similar to GET
usually, offers only meta-data

e.g., MIME type (Media Type) of a resource, last update,...

HTTP – Methods

PUT

updates a resource representation or, eventually,
creates a resource on the Web server

HTTP – Methods

POST

creates a resource, usually sending entities (data, actions) to the server

HTTP – Methods

POST

creates a resource, usually sending entities (data, actions) to the server

e.g., data entered into a Web form's fields

HTTP – Methods

DELETE

erases a resource – its representation –
from the server

HTTP – Methods

advanced

...and a few more

developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods

HTTP – Methods

Remark

traditionally, the Web browser
only permits the use of GET and POST methods
through Web forms

other methods can be used
only by constructing HTTP requests
through JavaScript

HTTP – Resource Representations

advanced

Character set encodings

ISO-8859-1

ISO-8859-2

KOI8-R

ISO-2022-JP

UTF-8

UTF-16 Little Endian

...

HTTP – Resource Representations

advanced

Representation formats

text

HTML, CSS, plain text, JavaScript code, XML document

or

binary

image, PDF document, multimedia resource, .gzip archive

HTTP – Resource Representations

Resource's content type

media types

HTTP – Header Fields (Attributes)

Content-Type

permits the transfer of any kind of data

Content-Type: `type/subtype`

HTTP – Header Fields (Attributes)

Content-Type

specified by Media Types – MIME
(Multipurpose Internet Mail Extensions)

denotes a set of primary content types
+ additional sub-types

initially, used in the e-mail context

HTTP – Header Fields (Attributes)

MIME types:



text indicates textual formats

text/plain – unformatted text

text/html – HTML (HyperText Markup Language) document

text/css – CSS (Cascading Style Sheets) resource

HTTP – Header Fields (Attributes)

MIME types:



image specifies graphical formats

image/gif – GIF (Graphics Interchange Format) images

image/jpeg – JPEG (Joint Picture Experts Group) photos

image/png – PNG (Portable Network Graphics) pictures

image/webp – WebP (Web Picture Format) images

image/avif – (animated) images optimized for the Web

image/svg+xml – SVG (Scalable Vector Graphics) illustrations

HTML `` – developer.mozilla.org/docs/Web/HTML/Element/img

HTTP – Header Fields (Attributes)

MIME types:



audio denotes audio content

audio/mpeg – resource encoded in MP3 format specification for audio data according to the MPEG (Motion Picture Experts Group) standard – tools.ietf.org/html/rfc3003

audio/ac3 – compressed audio resource conforming to AC-3 standard – www.atsc.org/standards/

HTML `<audio>` – developer.mozilla.org/docs/Web/HTML/Element/audio

HTTP – Header Fields (Attributes)

MIME types:

 **video** defines video content: animations, films


video/av1 – open format: Alliance for Open Media
aomedia.org/av1-features/

video/h265 – resource in High Efficiency Video Coding format
www.itu.int/rec/T-REC-H.265

HTML `<video>` – developer.mozilla.org/docs/Web/HTML/Element/video

HTTP – Header Fields (Attributes)

MIME types:

 **application** signifies formats that can be processed by applications on the client-side

application/epub+zip – EPUB e-book in ZIP format

application/javascript – JavaScript program

application/json – JSON (JavaScript Object Notation) data

application/octet-stream – stream of arbitrary bytes

application/senml+xml – XML data from sensors: RFC 8428

HTTP – Header Fields (Attributes)

MIME types:

 **multipart** used to transfer composed data

multipart/mixed – mixed content

multipart/alternative – alternative contents

HTTP – Header Fields (Attributes)

MIME types:

N. Freed *et al.*, *Media Types* (30 October 2025)

www.iana.org/assignments/media-types/media-types.xhtml

HTTP – Header Fields (Attributes)

Location

Location ":" "http(s)://" **authority** [":" **port**] [**abs_path**]

redirects the client to another resource representation
(HTTP redirect)

Location: <http://somewhere.info:8080/moved.html>

HTTP – Header Fields (Attributes)

Referer

denotes the URI of a Web resource that refers to the current resource

used to know the source of the requests to a given document (back-links) for analytics, logging, caching,...

HTTP – Header Fields (Attributes)

Host

specifies the target address
– IP or symbolic domain –
of the machine supposed to provide
a requested resource

HTTP – Header Fields (Attributes)

advanced

Other existing fields concern the following:

accepted content (content negotiation) – e.g., Accept
authentication & authorization – WWW-Authenticate Authorization
conditional access to resources – If-Match, If-Modified-Since,...
cache – Age, Cache-Control, Expires, ETag, etc.
proxy – Proxy-Authenticate, Proxy-Authorization, Via
message delivery (HTTP push) – Topic, TTL, Urgency
...and many many others

developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers

HTTP – Status

Informational (1xx)

100 Continue, 101 Switching Protocols,
102 Processing, 103 Early Hints

HTTP – Status

Success (2xx)

200 Ok, 201 Created, 202 Accepted,
204 No Content, 206 Partial Content,...

HTTP – Status

Redirection (3xx)

300 Multiple Choices, 301 Moved Permanently, 302 Found,
303 See Other, 304 Not Modified, 305 Use Proxy,...

HTTP – Status

Client Error (4xx)

400 Bad Request, 401 Unauthorized, 403 Forbidden,
405 Method Not Allowed, 408 Request Timeout, 410 Gone,
414 Request-URI Too Long, 415 Unsupported Media Type,
423 Locked, 429 Too Many Requests,...

HTTP – Status

Server Error (5xx)

500 Internal Server Error, 501 Not Implemented,
502 Bad Gateway, 503 Service Unavailable,
505 HTTP Version Not Supported, 508 Loop Detected,...

HTTP – Example of a Request

```
GET /web-development-foundations/ HTTP/1.1
Host: edu.info.uaic.ro
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:144.0)
          Gecko/20100101 Firefox/144.0
Accept: text/html,application/xhtml+xml,application/xml;
        q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Referer: https://profs.info.uaic.ro/andrei.panu/
```

HTTP – Example of a Response

HTTP/1.1 200 OK

Date: Thu, 30 Oct 2025 16:04:32 GMT

Server: Apache

Last-Modified: Mon, 27 Oct 2025 09:22:43 GMT

Content-Encoding: gzip

Content-Length: 1060

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: **text/html**; charset=UTF-8

header fields
(meta-data)

content

<!DOCTYPE html>

<html lang="ro">

..

..

</html>

NETFLIX Pagina principală Seriale Filme Noi și populare Lista mea

Inspector Console Debugger Performance Network Storage DOM Application

Filter URLs | Disable Cache | No Throttling

All HTML CSS JS XHR Fonts Images Media WS Other

Status	Method	Domain	File	Initiator	Type	Transferred	Size
304	GET	occ-0-3018-3466...	AAAABb89JNT487jNJ_mRF6AaGV3q9UWNzNxNKbsfF0suBLNc	img	jpeg	cached	41.04 KB
304	GET	occ-0-3018-3466...	AAAABeFa98bPMrT-JmX04T3-naWQ7ZH9kcmQRWHxaSSJN4C	img	jpeg	cached	25.01 KB
200	GET	codex.nflxext.com	none	script	js	3.96 KB	9.01 KB
200	GET	codex.nflxext.com	none	script	js	777.11 KB	2.62 MB
	GET	assets.nflxext.com	nficon2016.ico	FaviconLoader.jsm:1...	x-icon	16.56 KB (...)	16.56 KB
101	GET	push.prod.netflix.c...	ws	websocket	plain	175 B	0 B
Blocked	GET	ae.nflximg.net	adtech_iframe_target_05.html?data={"membership_status":"Cl	none:11 (subdocume...		Blocked B...	
200	POST	www.netflix.com	pathEvaluator?webp=false&drmSystem=widvine&isVolatileBillk	none:9 (xhr)	json	7.28 KB	28.72 KB
200	GET	www.netflix.com	probe?monotonic=false&device=web&iter=0	cadmium-playercore...	json	2.42 KB	1.12 KB
200	POST	www.netflix.com	pathEvaluator?webp=false&drmSystem=widvine&isVolatileBillk	none:9 (xhr)	json	2.48 KB	27 B
200	GET	nfrt-nrdp-nlb.prod...	probe?data=5120&device=web&ord=4&recipe=probnikNoFront	cadmium-playercore...	octet-stream	5.78 KB	5 KB
200	GET	nfrt-mobile-direct...	probe?data=5120&device=web&ord=0&recipe=probnikNoFront	cadmium-playercore...	octet-stream	5.79 KB	5 KB
200	GET	nfrt-mobile-nlb.pro...	probe?data=5120&device=web&ord=2&recipe=probnikNoFront	cadmium-playercore...	octet-stream	5.79 KB	5 KB
200	GET	nfrt-mobile-direct...	probe?data=5120&device=web&ord=1&recipe=probnikNoFront	cadmium-playercore...	octet-stream	5.79 KB	5 KB
200	GET	nfrt-nrdp-direct.pr...	probe?data=5120&device=web&ord=3&recipe=probnikNoFront	cadmium-playercore...	octet-stream	5.79 KB	5 KB
304	GET	occ-0-3018-3466...	AAAABZSqcXJDH3nWFZk9HJx5FMAabIXDeFHowYYVlcDQg1d...	none:6 (img)	jpeg	cached	35.59 KB
304	GET	occ-0-3018-3466...	AAAABYIET22cXmL7amS5zxBdaguxTnVQspnUjsapLJyKXeOQC...	none:6 (img)	jpeg	cached	18.54 KB
304	GET	occ-0-3018-3466...	AAAABWdYKiWuRs8MbYQ_6TiWe8exVZZECeva_6tP7OiNCLkrk...	none:6 (img)	jpeg	cached	16.78 KB

100 requests | 6.49 MB / 1.14 MB transferred | Finish: 2.09 min | DOMContentLoaded: 1.57 s | load: 5.04 s

advanced

inspecting HTTP requests made by the browser

HTTP – Web Server

fulfills multiple requests from the clients
using the HTTP protocol

HTTP – Web Server

fulfills multiple requests from the clients
using the HTTP protocol

- ⚠ each request is considered independent from others,
even though it comes from the same Web client
-> connection state is not kept – **stateless**



How can we develop
the back-end of Web applications?

Necessity

Dynamic generation – on the server –
of **representations** of resources
requested by clients

Solutions

CGI – Common Gateway Interface

Web application servers

Web frameworks

CGI

advanced

Language-independent programming interface
facilitating the interaction between clients and
programs invoked on the Web server

de facto standard

RFC 3875 – datatracker.ietf.org/doc/html/rfc3875
www.w3.org/CGI/

CGI

advanced

CGI scripts can be written in any language
available on the server

interpreted languages

bash, Perl, Python, Ruby,...

compiled languages

C, C++, Rust,...

CGI – Programming

advanced

Any CGI program will write data
– the representation of a Web resource –
at standard output (stdout)



CGI – Programming

advanced

To denote the type of the generated representation,
HTTP headers are used – MIME (Media Types)

examples:

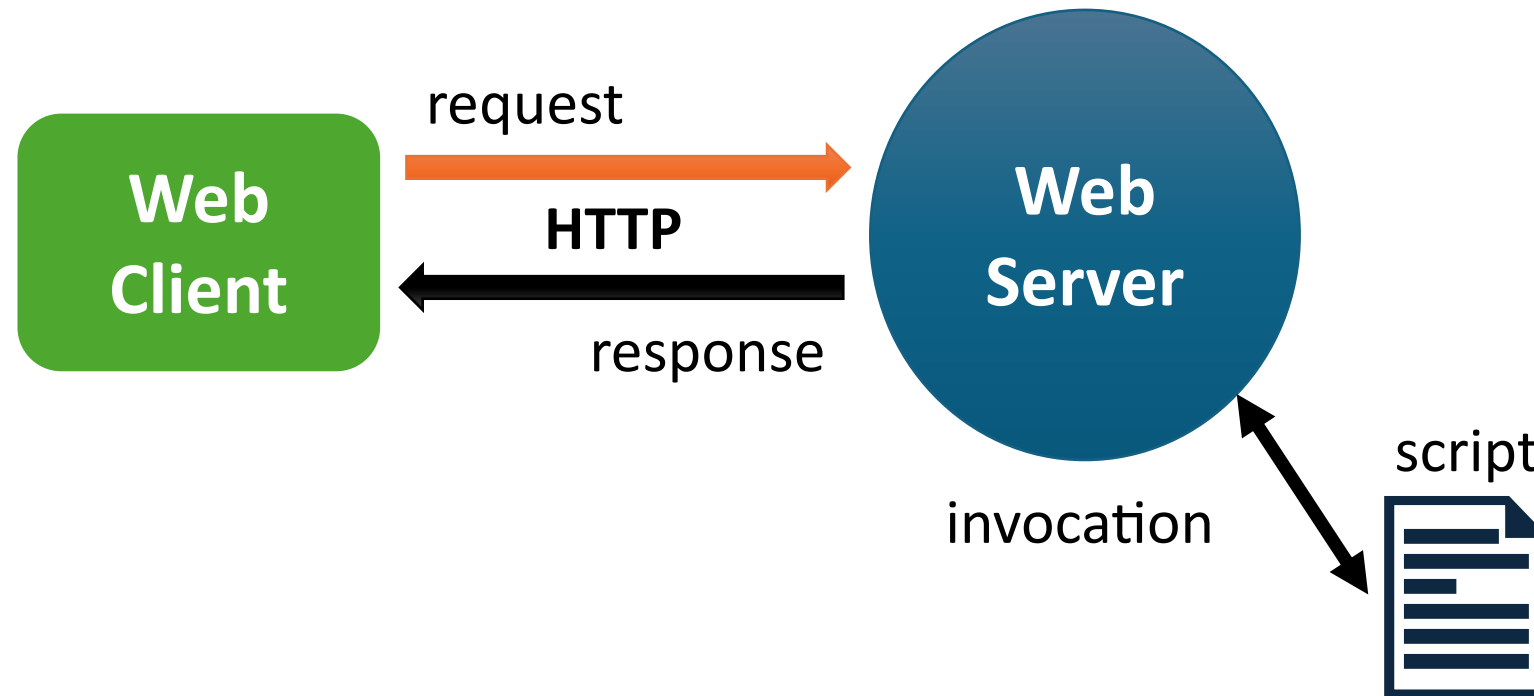
Content-type: text/html

Content-type: image/webp

CGI – Programming

advanced

The interaction between the client and Web server



CGI – Variables

advanced

A CGI script has access to environment variables associated to the request sent to the CGI program:

REQUEST_METHOD – HTTP method (GET, POST,...)

QUERY_STRING – data transmitted to the client

REMOTE_HOST, REMOTE_ADDR – client address

CONTENT_TYPE – content type as MIME (Media Type)

CONTENT_LENGTH – content length in bytes

CGI – Variables

advanced

Additional variables
usually, generated by the Web server:

HTTP_ACCEPT – MIME types accepted by client (browser)

HTTP_COOKIE – data about cookies

HTTP_HOST – information regarding the host (client)

HTTP_USER_AGENT – information about the client

...and others

CGI – Variables

Obtaining the values of the system's environment variables by invoking, through **GET**, the variable.cgi script stored on the Web server (the program must have read and execution permissions)

```
#!/bin/bash

# Setting content type
echo "Content-type: text/plain";
echo

# Executing 'set' command in Linux
# to display the environment variables
set
```

```
https://profs.info.uaic.ro/~busaco/cgi/bash/variabile.cgi?date_de_intrare
GATEWAY_INTERFACE=CGI/1.1
GROUPS=()
HOSTNAME=profs.info.uaic.ro
HOSTTYPE=x86_64
HTTPS=on
HTTP_ACCEPT='text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'
HTTP_ACCEPT_ENCODING='gzip, deflate, br'
HTTP_ACCEPT_LANGUAGE='en-GB,en;q=0.5'
HTTP_CONNECTION=keep-alive
HTTP_DNT=1
HTTP_HOST=profs.info.uaic.ro
HTTP_SEC_GPC=1
HTTP_UPGRADE_INSECURE_REQUESTS=1
HTTP_USER_AGENT='Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0) Gecko/20100101 Firefox
IFS=$' \t\n'
MACHTYPE=x86_64-redhat-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/bin:/usr/bin:/bin
PIPESTATUS=( [0]="0" )
PPID=25485
PS4='+ '
PWD=/home/thor/profs/busaco/html/cgi/bash
QUERY_STRING=date_de_intrare
REMOTE_ADDR=109.102.38.36
REMOTE_PORT=50606
REQUEST_METHOD=GET
REQUEST_SCHEME=https
REQUEST_URI='/~busaco/cgi/bash/variabile.cgi?date_de_intrare'
SCRIPT_FILENAME=/thor/profs/busaco/public_html/cgi/bash/variabile.cgi
SCRIPT_NAME=/~busaco/cgi/bash/variabile.cgi
SERVER_ADDR=85.122.23.20
SERVER_ADMIN=pimi@info.uaic.ro
SERVER_NAME=profs.info.uaic.ro
SERVER_PORT=443
SERVER_PROTOCOL=HTTP/1.1
SERVER_SIGNATURE=
SERVER_SOFTWARE='Apache/2.4.35 (IUS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/7.2.34'
```

the result obtained
by the Web client

advanced

```
GATEWAY_INTERFACE=CGI/1.1
GROUPS=()
HOSTNAME=profs.info.uaic.ro
HOSTTYPE=x86_64
HTTPS=on
HTTP_ACCEPT='text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'
HTTP_ACCEPT_ENCODING='gzip, deflate, br'
HTTP_ACCEPT_LANGUAGE='en-GB,en;q=0.5'
HTTP_CONNECTION=keep-alive
HTTP_DNT=1
HTTP_HOST=profs.info.uaic.ro
HTTP_SEC_GPC=1
HTTP_UPGRADE_INSECURE_REQUESTS=1
HTTP_USER_AGENT='Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0) Gecko/20100101 Firefox
IFS=$' \t\n'
MACHTYPE=x86_64-redhat-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/bin:/usr/bin:/bin
PIPESTATUS=( [0]="0" )
PPID=25485
PS4='+ '
PWD=/home/thor/profs/busaco/html/cgi/bash
QUERY_STRING=date_de_intrare
REMOTE_ADDR=109.102.38.36
REMOTE_PORT=50606
REQUEST_METHOD=GET
REQUEST_SCHEME=https
REQUEST_URI='/~busaco/cgi/bash/variabile.cgi?date_de_intrare'
SCRIPT_FILENAME=/thor/profs/busaco/public_html/cgi/bash/variabile.cgi
SCRIPT_NAME=/~busaco/cgi/bash/variabile.cgi
SERVER_ADDR=85.122.23.20
SERVER_ADMIN=pimi@info.uaic.ro
SERVER_NAME=profs.info.uaic.ro
SERVER_PORT=443
SERVER_PROTOCOL=HTTP/1.1
SERVER_SIGNATURE=
SERVER_SOFTWARE='Apache/2.4.35 (IUS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/7.2.34'
```

the result obtained
by the Web client

CGI programs written in C, bash,
Python generating
the same HTML content

```
/* hello.c
   (compile with gcc hello.c -o hello.cgi)
*/
#include <stdio.h>

int main() {
    int msgs; /* number of messages */

    printf ("Content-type: text/html\n\n");

    for (msgs = 0; msgs < 10; msgs++) {
        printf ("<p>Hello, world!</p>");
    }

    return 0;
}
```

```
#!/bin/bash
# hello.sh.cgi

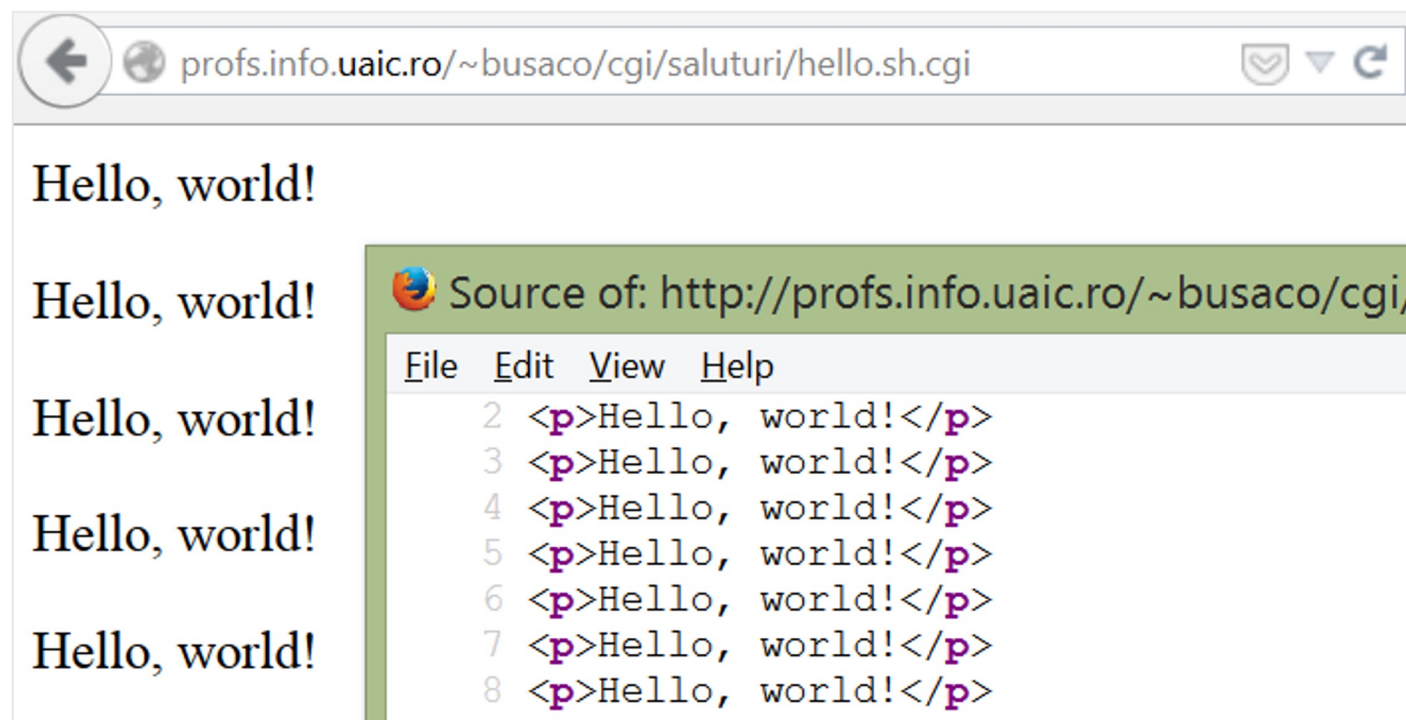
echo "Content-type: text/html"
echo

MESSAGES=0
while [ $MESSAGES -lt 10 ]
do
    echo "<p>Hello, world!</p>"
    let MESSAGES=MESSAGES+1
done
```

```
#!/usr/bin/python
# hello.py.cgi

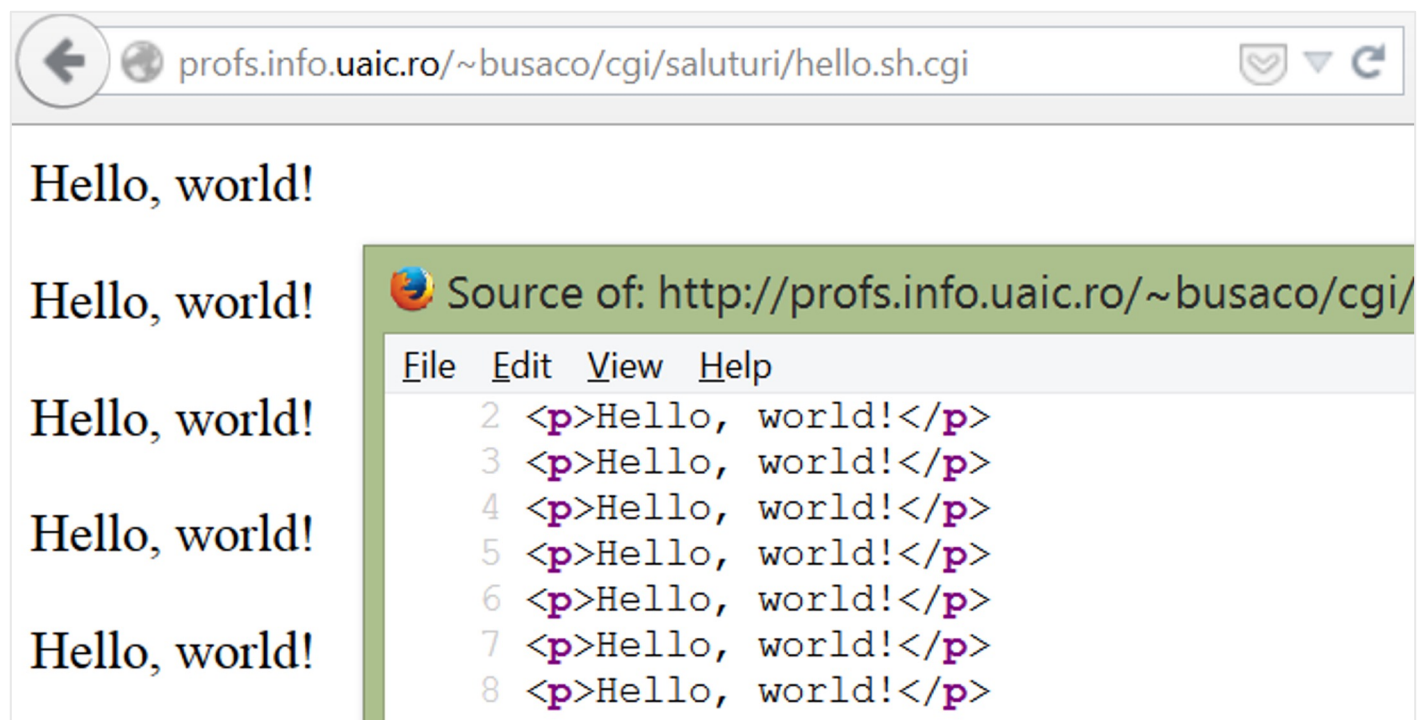
print "Content-type: text/html\n"

for messages in range (0, 10):
    print "<p>Hello, world!</p>"
```



the client (e.g., browser) receives the representation as response – here, the HTML page – generated by the CGI program invoked by the Web server

this representation is processed and, eventually, displayed in a (zone of a) browser window



by experimenting other MIME types,
the browser displays the following:

```
<p>Hello, world!</p>  
<p>Hello, world!</p>  
<p>Hello, world!</p>  
<p>Hello, world!</p>  
<p>Hello, world!</p>
```

Content-type: text/plain

```
XML Parsing Error: junk after document element  
Location: http://profs.info.uaic.ro/~busaco/cgi/hello  
Line Number 2, Column 1:  
  
<p>Hello, world!</p>  
^
```

Content-type: text/xml

CGI – Invocation

```
<form action="https://somewhere.info/cgi/get-max.cgi"
        method="GET">
  <p>Enter two numbers :
    <input type="text" name="no1" />
    <input type="text" name="no2" />
  </p>
  <input type="submit" value="Compute maximum" />
</form>
```

invocation from an interactive Web form
in this case, using the **GET** method

https://profs.info.uaic.ro/~busaco/cgi/max.html

Enter two numbers:

Compute maximum

special URL
in GET case

profs.info.uaic.ro/~busaco/cgi/get-max.cgi?no1=7&no2=4

Maximum of two numbers

$\max(7, 4) = 7$

CGI – Invocation

For each form field, a **field_name=value** pair
– delimited by **&** – is generated and added to the URL
of the CGI script to be invoked on server

somewhere.info/cgi/get-max.cgi?no1=7&no2=4

CGI – Invocation

The server will invoke the CGI script
passing the data received from the client
via **environment variables** – usually, in **GET** case
or
at **standard input (stdin)** – if **POST** is used

CGI – Invocation

advanced

Data processing when **GET** method is used

data available in **QUERY_STRING** environment variable

CGI – Invocation

advanced

Data processing when **POST** method is used

data read from stdin, the length in bytes being specified by **CONTENT_LENGTH** variable

CGI – Invocation

Data processing – **GET** and/or **POST**

in case of application servers or frameworks,
data is encapsulated into specific structures/types

ASP.NET (C# et. al) – **HttpRequest** class

Node.js (JavaScript) – **http.ClientRequest**

PHP – associative arrays: **\$_GET[]** **\$_POST[]** **\$_REQUEST[]**

Play (Java, Scala) – **play.api.mvc.Request**

Python – class **cgi.FieldStorage**

GET vs. POST

GET method is used to get representations of the resources requested from the Web server

e.g., HTML documents, JPEG, PNG, or WebP images, SVG illustrations (XML), PDF resources, ZIP archives, etc.

 **the server state should not be modified**

GET vs. POST

GET method is used to get representations of the resources requested from the Web server

obtaining data with GET, the user can set a bookmark for further accesses to the Web resource (by using the URL of the generated representation)

e.g., getpocket.com/search/?q=web+dev&st=saves&ft=videos

GET vs. POST

POST method is used when the data transmitted to the server is large (e.g., upload of file content) or sensitive – typical example: passwords

GET vs. POST

POST method is used when the data transmitted to the server is large (e.g., upload of file content) or sensitive – typical example: passwords

also, when the script invocation can produce a state change on the server:
adding a record, altering a file,...

CGI – Support

advanced

The Web server should support
CGI script invocation

example:

Apache HTTP Server provides the **mod_cgi** module

Web Application Server

purpose:

optimizing the development processes
of complex Web applications

Web Application Server

advanced

purpose:

optimizing the development processes
of complex Web applications

could encourage or impose an architectural approach for Web
application development – e.g., MVC or variants

Framework

facilitates complex Web application development,
by simplifying usual operations
(e.g., access to databases, caching, code generation,
session management, access control) and/or
encouraging the source code reuse



Summary