Web Technologies

Dr. Angel J. Lopez

HTTP

HyperText Transfer Protocol

HTTP

- Evolution of HTTP
- Uniform Resource Identifier (URI)
    - Uniform Resource Locator (URL)
    - Uniform Resource Names (URN)
- Basic operations
- Message Format and Status Codes

HTTP (HyperText Transfer Protocol)

- HTTP is a request/response standard of a client and a server

- Typically, an HTTP client initiates a request

- Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs)

Evolution of HTTP

HTTP/0.9 – The one-line protocol

- The initial version of HTTP had no version number
- HTTP/0.9 is extremely simple
- **Requests** consist of a <u>single line</u> and start with the only possible method **GET** followed by the **path** to the resource

```
1   GET /mypage.html
```

HTTP/0.9 – The one-line protocol

- **Requests** consist of a <u>single line</u> and start with the only possible method **GET** followed by the **path** to the resource

```
1   GET /mypage.html
```

- The **response** is extremely simple too: it only consisted of the file itself.

```
1   <HTML>
2   A very simple HTML page
3   </HTML>
```

HTTP/0.9 – The one-line protocol

HTTP/0.9 is extremely simple:

- **no HTTP headers**, meaning that <u>only HTML files</u> could be transmitted, but no other type of documents.

- **no status or error codes:** in case of a problem, a specific HTML file was send back with the description of the problem contained in it, for human consumption.

HTTP/1.0 – Building extensibility

- **Versioning information** is now sent within each request
  - `HTTP/1.0`

- A **status code** line is also sent at the beginning of the response

- **HTTP headers** (request / response)
  - flexible and extensible
  - other documents than plain HTML files (`Content-Type` header).

```
1   GET /mypage.html HTTP/1.0
2   User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
3
4   200 OK
5   Date: Tue, 15 Nov 1994 08:12:31 GMT
6   Server: CERN/3.0 libwww/2.17
7   Content-Type: text/html
8   <HTML>
9   A page with an image
10    <IMG SRC="/myimage.gif">
11  </HTML>
```

HTTP/1.0 – Building extensibility

```
1    GET /mypage.html HTTP/1.0          ← Versioning information
2    User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
3
4    200 OK                              ← Status Code
5    Date: Tue, 15 Nov 1994 08:12:31 GMT
6    Server: CERN/3.0 libwww/2.17
7    Content-Type: text/html
8    <HTML>
9    A page with an image
10      <IMG SRC="/myimage.gif">
11   </HTML>
```

HTTP Headers (lines 5–7)

HTTP/1.0 – Building extensibility

```
GET /myimage.gif HTTP/1.0          ←———  **Versioning information**
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK                             ←————————  **Status Code**
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif             ←———  **Image file**
(image content)
```

HTTP Headers

HTTP/1.0 – Building extensibility

- **try-and-see** approach (1991-1995)

- A lot of interoperability problems

- Definition of HTTP/1.0 (RFC 1945, Nov 1996)
  - RFC 1945: https://tools.ietf.org/html/rfc1945
  - No official standard.

HTTP/1.1 – The standardized protocol

- **First standardized version** of HTTP
  - HTTP/1.1 (January 1997)
  - RFC 2068 (https://tools.ietf.org/html/rfc2068)
- A **connection** can be **reused**
- **Pipelining** (allow to send a second request before the answer for the first one is fully transmitted)
- Chunked responses
- Cache control mechanisms
- Content negotiation

HTTP/1.1 – The standardized protocol

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header

200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

(content)
```

HTTP/1.1 – The standardized protocol

Extensions (over 15 years):

- Using HTTP for secure transmissions
- Using HTTP for complex applications
  - WebDAV (1996)
  - Representational state transfer or REST (2000)

HTTP/2 – A protocol for greater performance

- **Binary protocol** (no longer be read and created manually)
- **Multiplexed protocol** (parallel requests can be handled)
- It compresses headers (remove duplication and overhead of data transmitted)
- Server push (It allows a server to populate data in a client cache)
- Officially standardized, in May 2015

URI

Uniform Resource Identifier

URL (Uniform Resource Locator)

- A URL is nothing more than the address of a given unique resource on the Web

https://developer.mozilla.org
https://developer.mozilla.org/en-US/docs/Learn/
https://developer.mozilla.org/en-US/search?q=URL

URL Anatomy

- A URL is composed of different parts, some mandatory and others optional.

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

↳ *Protocol*

A protocol is a set method for exchanging or transferring data around a computer network, e.g http, https

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

→ *Protocol*

tp://www.example.com:80/path/to/my

→ *Domain Name*

It indicates which Web server is being requested

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

↳ *Protocol*

tp://www.example.com:80/path/to/my

↳ *Domain Name*

com:80/path/to/myfile.html?key1=valu

↳ *Port*   It indicates the technical "gate" used to access the resources on the web server.

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

→ *Protocol*

tp://www.example.com:80/path/to/my

→ *Domain Name*

com:80/path/to/myfile.html?key1=valu

→ *Port*

n:80/path/to/myfile.html?key1=value1

→ *Path to the file*

It is the path to the resource on the Web server

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/
↳ *Protocol*

tp:// **www.example.com** :80/path/to/my
↳ *Domain Name*

com **:80** /path/to/myfile.html?key1=valu
↳ *Port*

n:80 **/path/to/myfile.html** ?key1=value1
↳ *Path to the file*

A list of key/value pairs (separated with the & symbol) provided to the Web server

html **?key1=value1&key2=value2** #Some
↳ *Parameters*

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

→ *Protocol*

www.example.com:80/path/to/my

→ *Domain Name*

:80/path/to/myfile.html?key1=valu

→ *Port*

/path/to/myfile.html?key1=value1

→ *Path to the file*

?key1=value1&key2=value2#Some

→ *Parameters*

An anchor represents a sort of "bookmark" inside the resource #SomewhereInTheDocument

→ *Anchor*

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80/path/

Protocol

tp://www.example.com:80/path/to/my

Domain Name

com:80/path/to/myfile.html?key1=valu

Port

n:80/path/to/myfile.html?key1=value1&

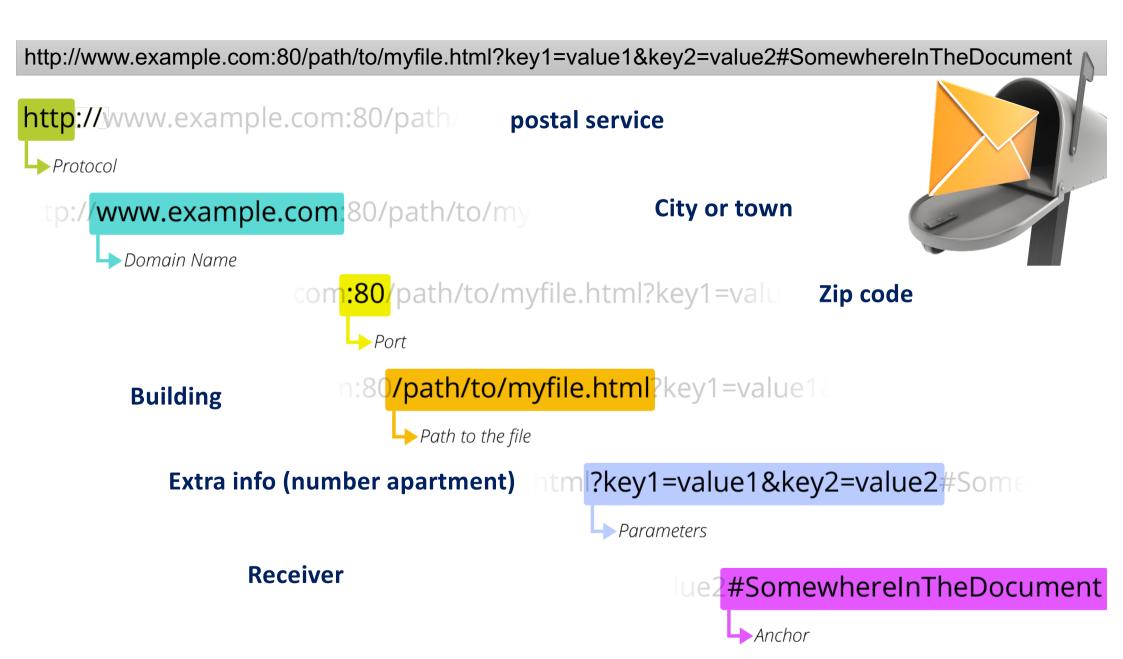Path to the file

html?key1=value1&key2=value2#Some

Parameters

lue2#SomewhereInTheDocument

Anchor

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

http:// www.example.com:80/path/

http: //www.example.com:80/path/

→ *Protocol*

postal service

tp:// **www.example.com** 80/path/to/my

→ *Domain Name*

City or town

com **:80** /path/to/myfile.html?key1=valu

→ *Port*

Zip code

n:80 **/path/to/myfile.html** ?key1=value1

Building

→ *Path to the file*

Extra info (number apartment) html **?key1=value1&key2=value2** #Some

→ *Parameters*

Receiver

lue2 **#SomewhereInTheDocument**

→ *Anchor*

Request Methods

- HTTP defines methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified **resource**.
  - HEAD
  - **GET**
  - **POST**
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT
  - PATCH

Safe methods

- HEAD, GET, OPTIONS and TRACE are defined as safe (no side effects).
- POST, PUT and DELETE are intended for actions which may cause side effects either on the server.

Status codes

- The first line of the HTTP response is called the status line
- The way the user agent handles the response primarily depends on the code and secondarily on the response headers
  - Success: 2xx
  - Redirection: 3xx
  - Client-Side Error: 4xx
  - Server-Side Error: 5xx

Practice

Request the following resource using HTTP:

https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html

Check for:

• Method

• Versioning information

• Status code

HTTP session state

- HTTP is a stateless protocol.
- Hosts do not need to retain information
- about users between requests.
- Statelessness is a scalability property.
- For example, when a host needs to customize the content of a website for a user. Solution:
  - Cookies
  - Sessions
  - Hidden variables (when the current page is a form)
  - URL encoded parameters

Cookie

- Cookie is a small piece of text stored on a user's computer by a web browser.

- A cookie consists of one or more name- value pairs containing bits of information such as user preferences.

- A cookie can be used for:
    - authenticating,
    - session tracking, and
    - remembering specific information about users