CTIS 256

Introduction to Backend Development

Notes # 1

HTTP

Serkan GENÇ

Introduction

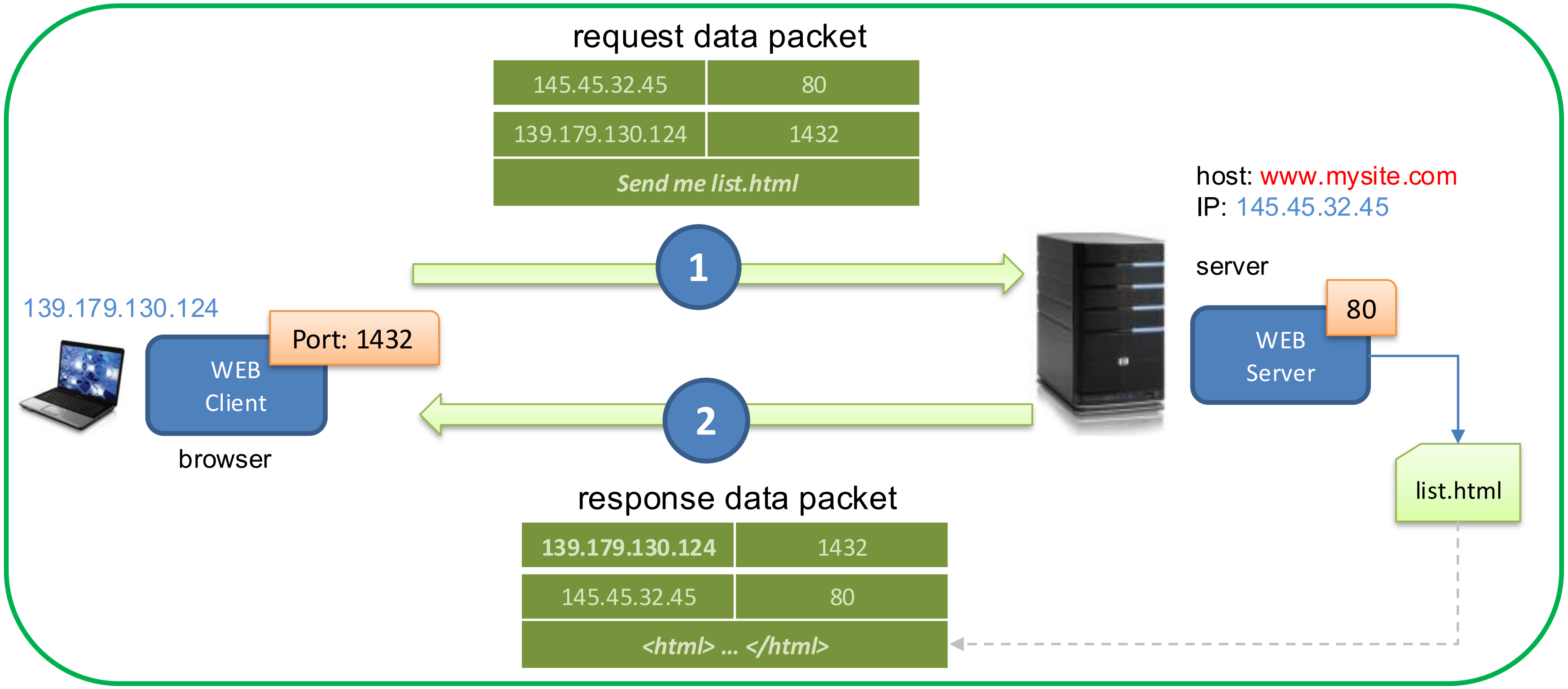
- Aim: Understand core backend concepts *(http protocol, form processing, validation, CRUD, session, REST API, security, etc.)* to develop web-based applications using nodejs and mySQL (DBMS).
- Web Application is a program accessed through a web browser.
- World Wide Web is a system of interlinked hypertext documents or applications accessed via the internet using web browsers. In WWW, there are two kinds of softwares; Web client, and Web Server.
- Web Server is a program that is responsible for distributing resources to the outside world. Web Client (Browser) is a program that requests files/resources from a Web Server.
- Why Web-based Application Development?
  - All application files are stored in server-side
  - **Easy to maintain**: bug fixes, adding new features, the same version for all users
  - **Platform independent**: it works in any platforms (unix,windows, iphone,etc). Develop application once for all platforms, so, it is time and cost effective.
  - **Advanced security :** no way to crack, and prevention of stealing know-how
  - **Connectivity**: one can easily connects to the system via browser and uses it immediately, no need to install, or setup.
  - **Flexible Licenses**: renting software for a specific period of time such as one month

Data Communication with TCP/IP

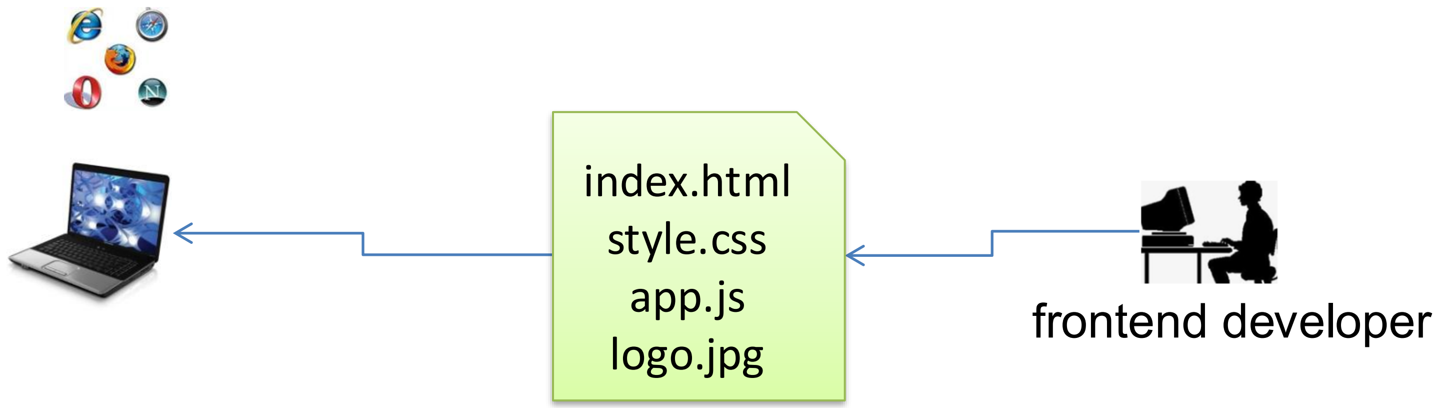
IP address: *used to uniquely identify devices on a network, allowing them to communicate with each other over the Internet.*
Port Number: *a numerical identifier in networking used to specify particular processes (program) or services on a device.*
TCP/IP: *a set of rules that allows devices to communicate over the Internet, ensuring data is sent and received correctly.*

Dst IP	Dst Port #
Src IP	Src Port #
Payload (high level protocol)	

tcp/ip packet format *(simplified)*



Static Content

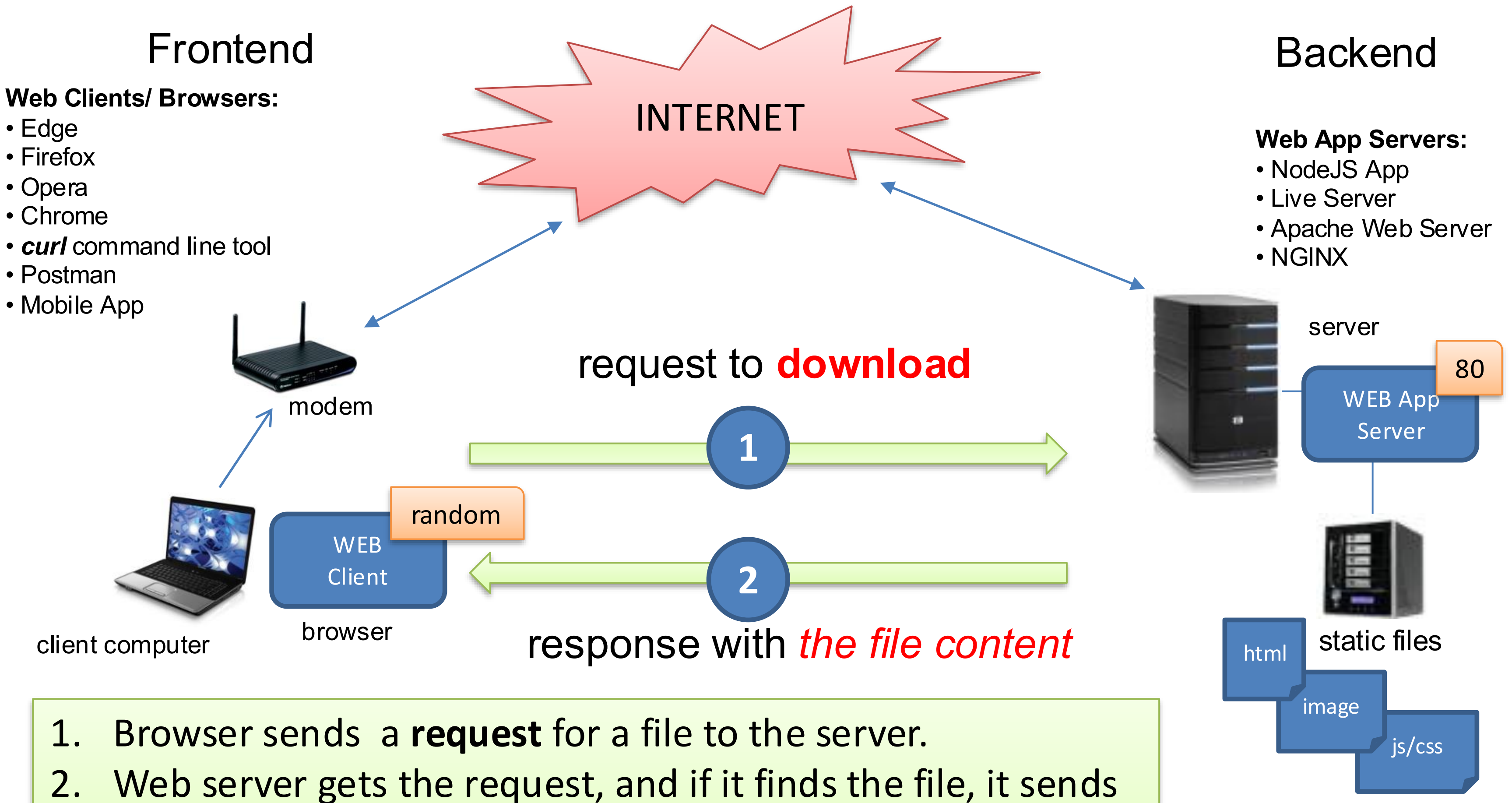


- *content that remains the same for every user*
- *does not change in response to user interactions*
- *html files, css files, image files, javascript files, pdf files are static.*

Overall picture of World Wide Web Architecture

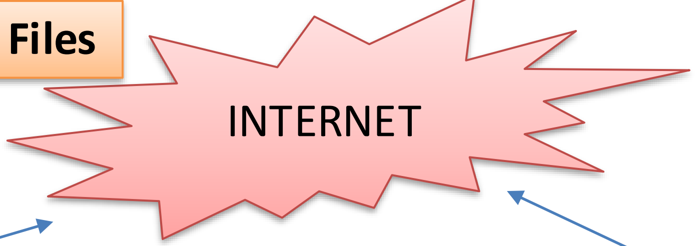
(Client-Server Architecture)

Solution to Delivery Problem

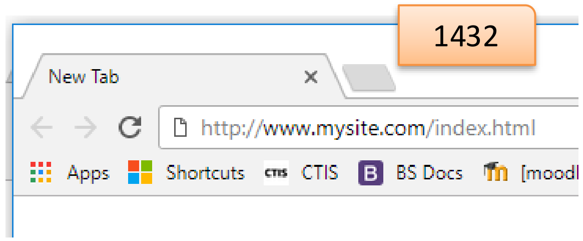


1. Browser sends a **request** for a file to the server.
2. Web server gets the request, and if it finds the file, it sends the content of the requested file back to the client. Otherwise, it sends the error code. This is called **response**.
3. A request/response pair is called a **transaction**.

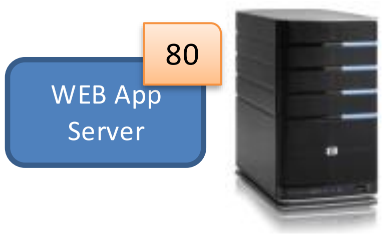
WWW Architecture – **Static Files**



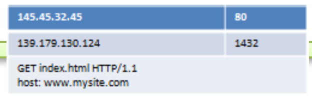
139.179.130.124



host: www.mysite.com
IP: 145.45.32.45



3 Browser sends an **HTTP Request** to the server.



145.45.32.45	80
139.179.130.124	1432
GET index.html HTTP/1.1 host: www.mysite.com	

4 Web Server gets HTTP request packet
Checks out the content of the packet
Finds the requested file in its storage.

5 Prepares HTTP response packet and sends back to client.

139.179.130.124	1432
145.45.32.45	80
HTTP/1.1 200 OK Date: Sat, 15 Jan 2025 14:37:12 GMT Server: Apache 1.45 Content-Type: text/HTML Content-Length: 29 <html> <h1>hello</h1> </html>	

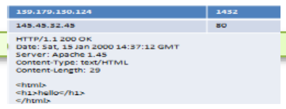
1 Launch a web client(browser)
Type the web address of the server (URL) into location bar (www.mysite.com) and press enter. Browser retrieves IP address from DNS server.

2 Browser prepares and sends a network packet called HTTP request packet *(root folder / sign if filename is not given, means "default file" selected by server)*

145.45.32.45 (dest IP)	80 (dest Port)
139.179.130.124 (src IP)	1432 (src Port)
GET index.html HTTP/1.1 host: www.mysite.com	

HTTP Request format

TCP/IP Packet (simplified)



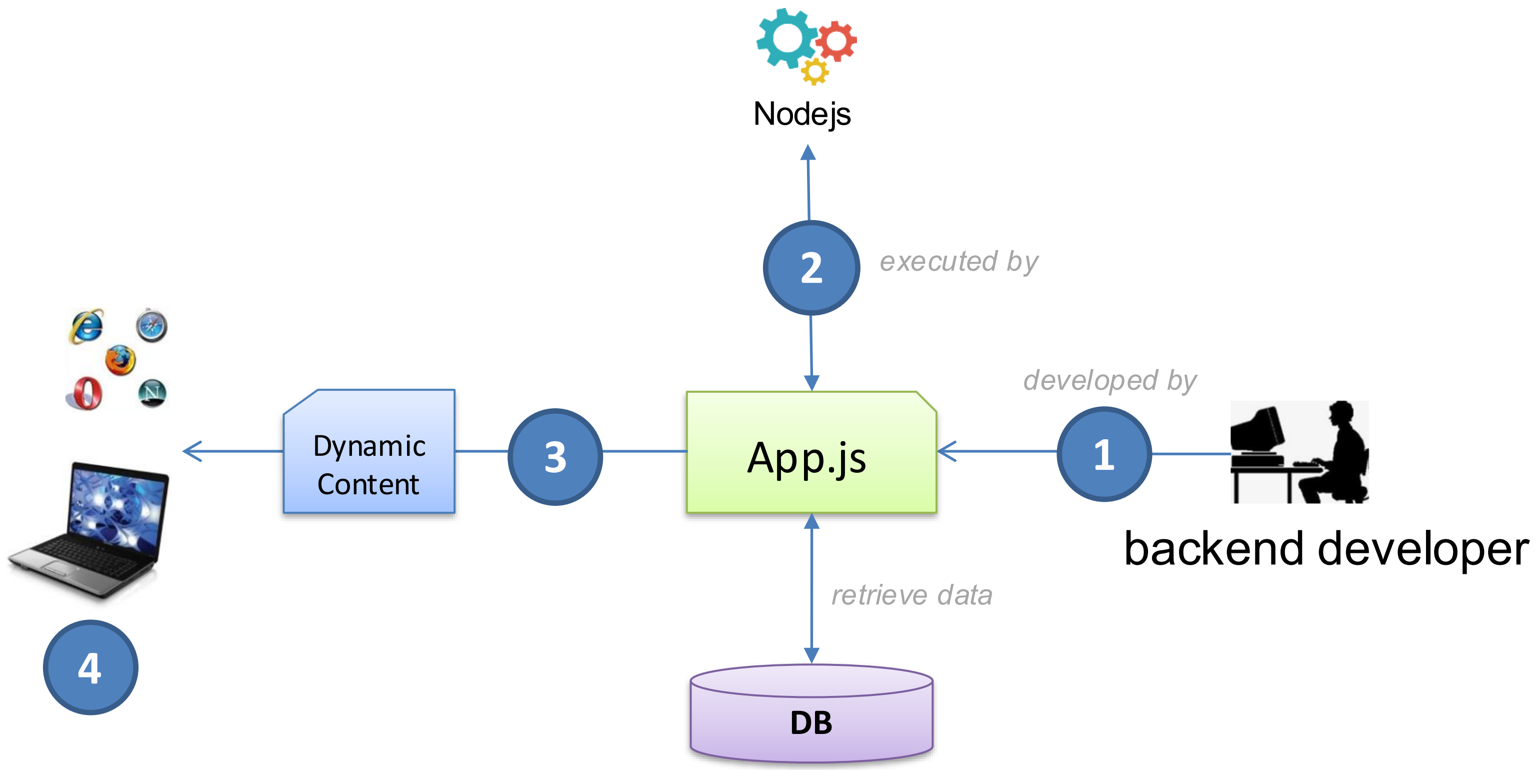
139.179.130.124	1432
145.45.32.45	80
HTTP/1.1 200 OK Date: Sat, 15 Jan 2000 14:37:12 GMT Server: Apache 1.45 Content-Type: text/HTML Content-Length: 29 <html> <h1>hello</h1> </html>	

7 Browser gets response packet, takes out HTTP part, and renders HTML codes.

6 Web Server sends an **HTTP response** packet to the client (browser).

80 : standard port address for Web Servers

Dynamic Content

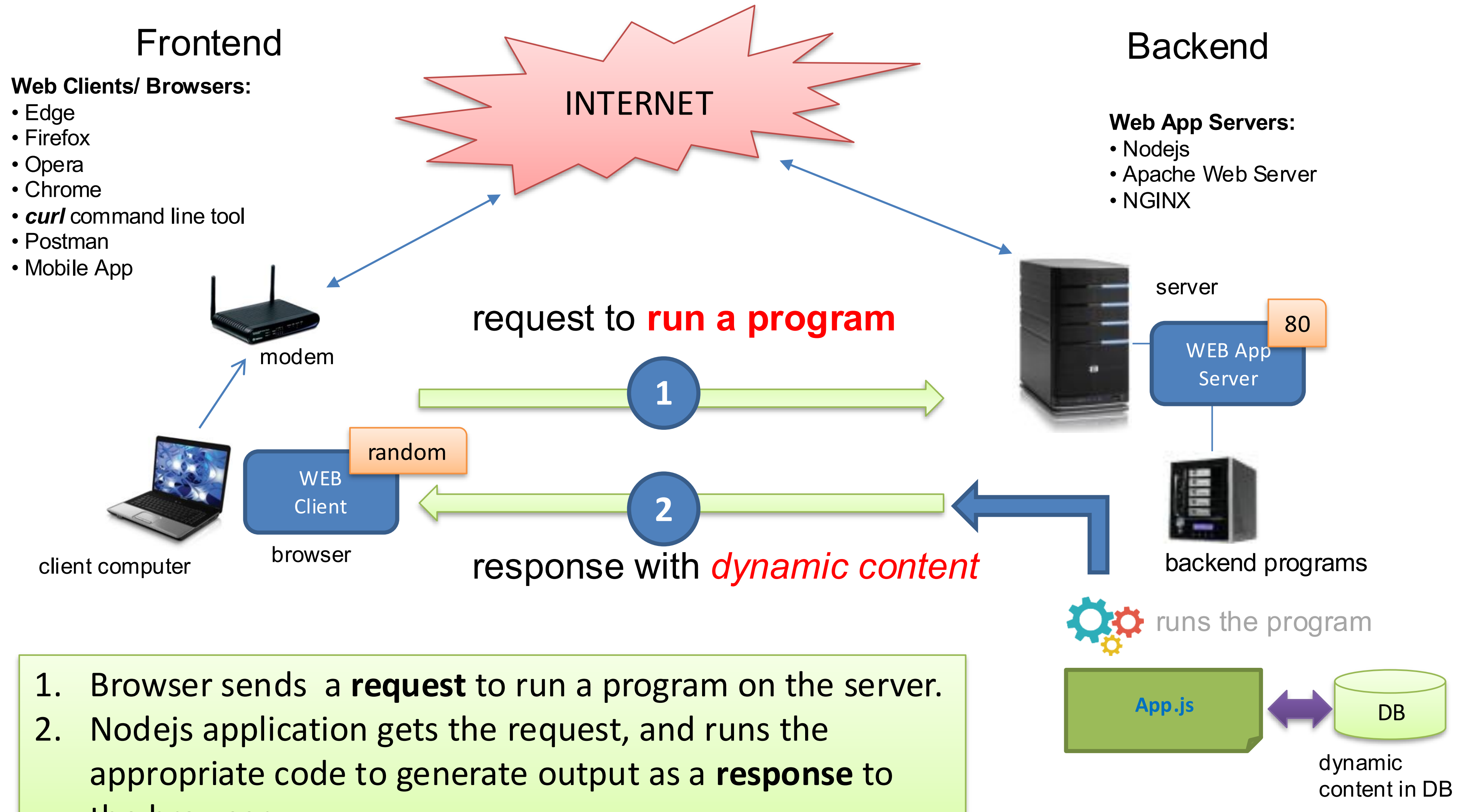


1. *A Backend developer writes a server-side program (app.js), called backend program.*
2. *Upon receiving a request, the application processes the incoming http request.*
3. *The program retrieves data from the database, and generates dynamic content based on that data.*
4. *The browser renders the dynamic content prepared by app.js. It can see only the output of the program, and cannot access the source code.*

Overall picture of World Wide Web Architecture

(Client-Server Architecture)

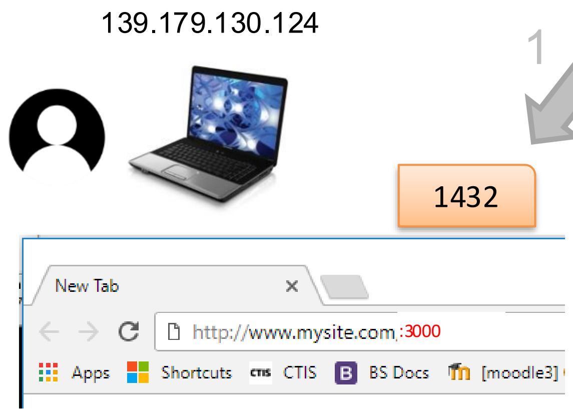
Solution to Dynamic Content



1. Browser sends a **request** to run a program on the server.
2. Nodejs application gets the request, and runs the appropriate code to generate output as a **response** to the browser.

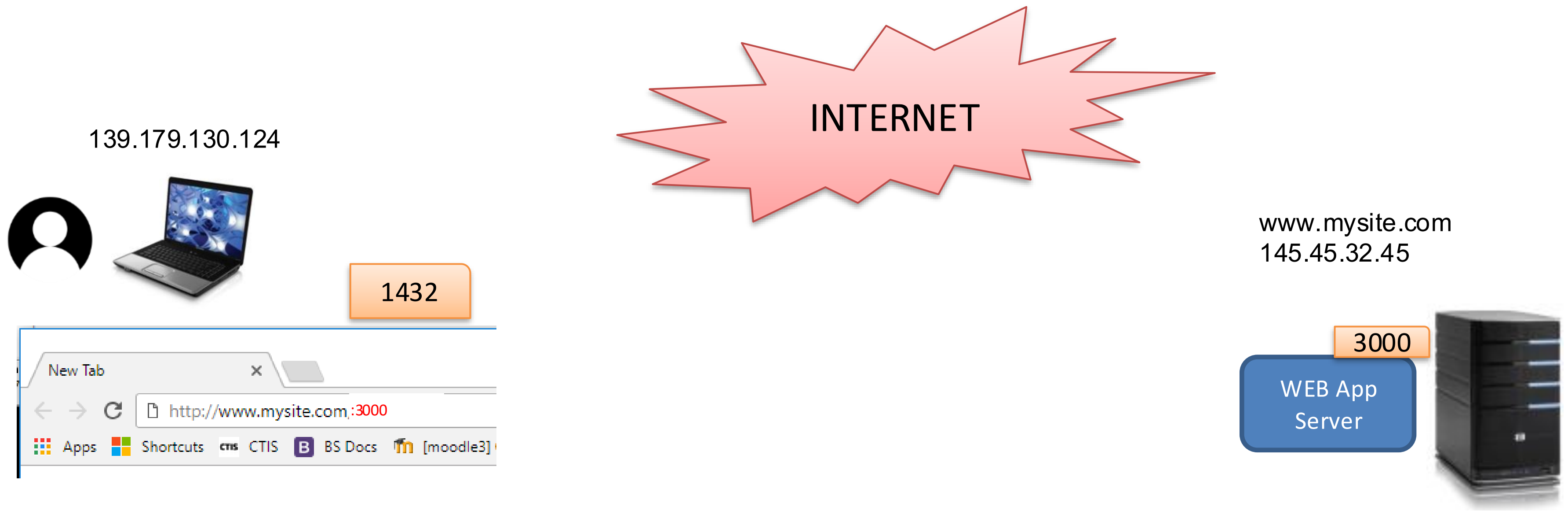
WWW Architecture – **Dynamic Content**

Step by Step



Open up a browser.
Operating System (windows, macos, linux) assigns a random port number

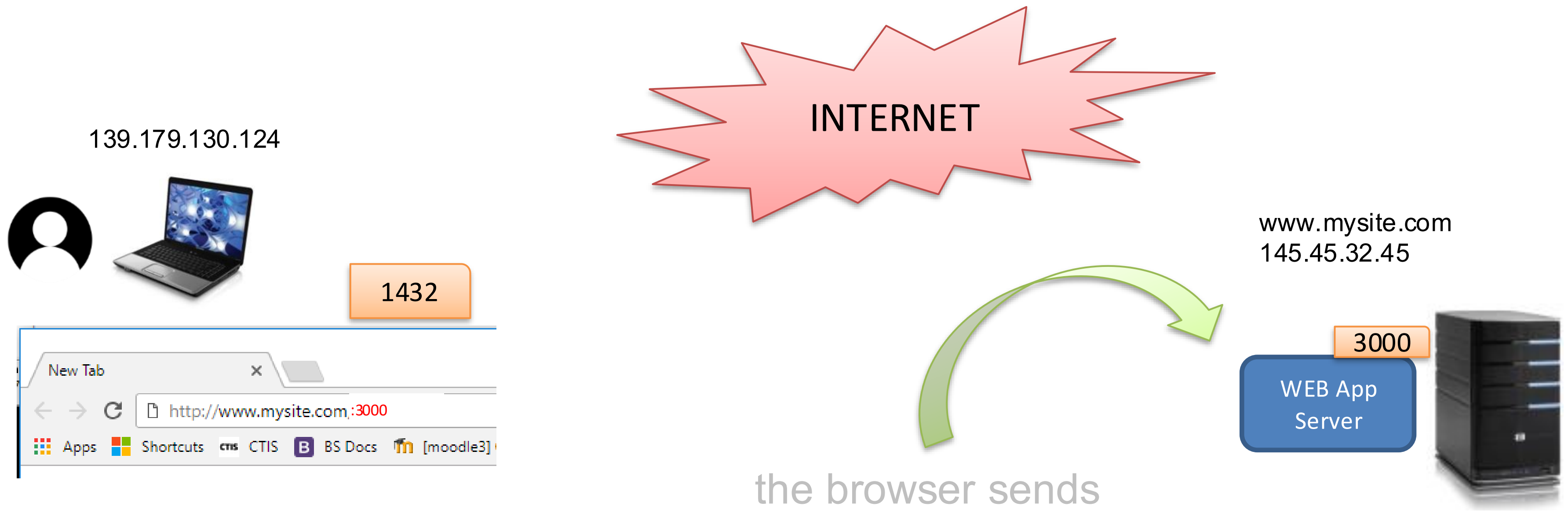
a user types URL address of the resource (file, program), and press Enter key.



the browser prepares http request packet

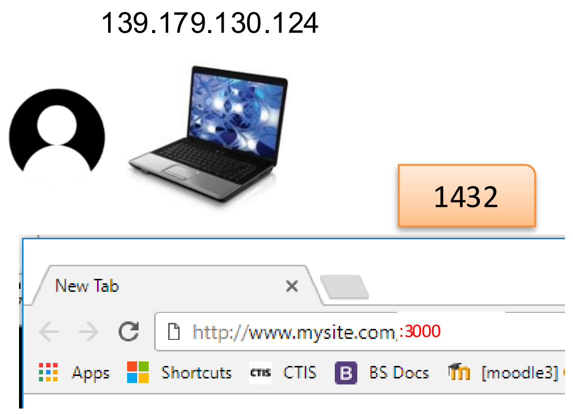
TCP/IP Packet

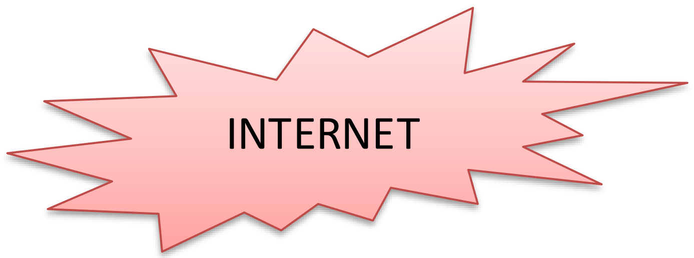
145.45.32.45 (dest IP)	3000 (dest)
139.179.130.124 (src IP)	1432 (src Port)
GET / HTTP/1.1 host: www.mysite.com	

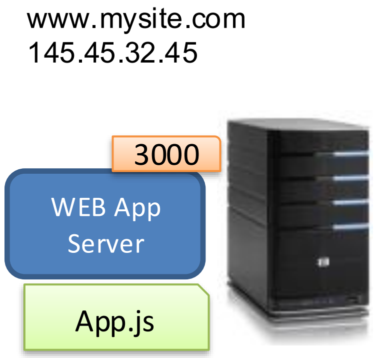


the browser sends
http request packet

145.45.32.45 (dest IP)	3000 (dest)
139.179.130.124 (src IP)	1432 (src Port)
GET / HTTP/1.1 host: www.mysite.com	

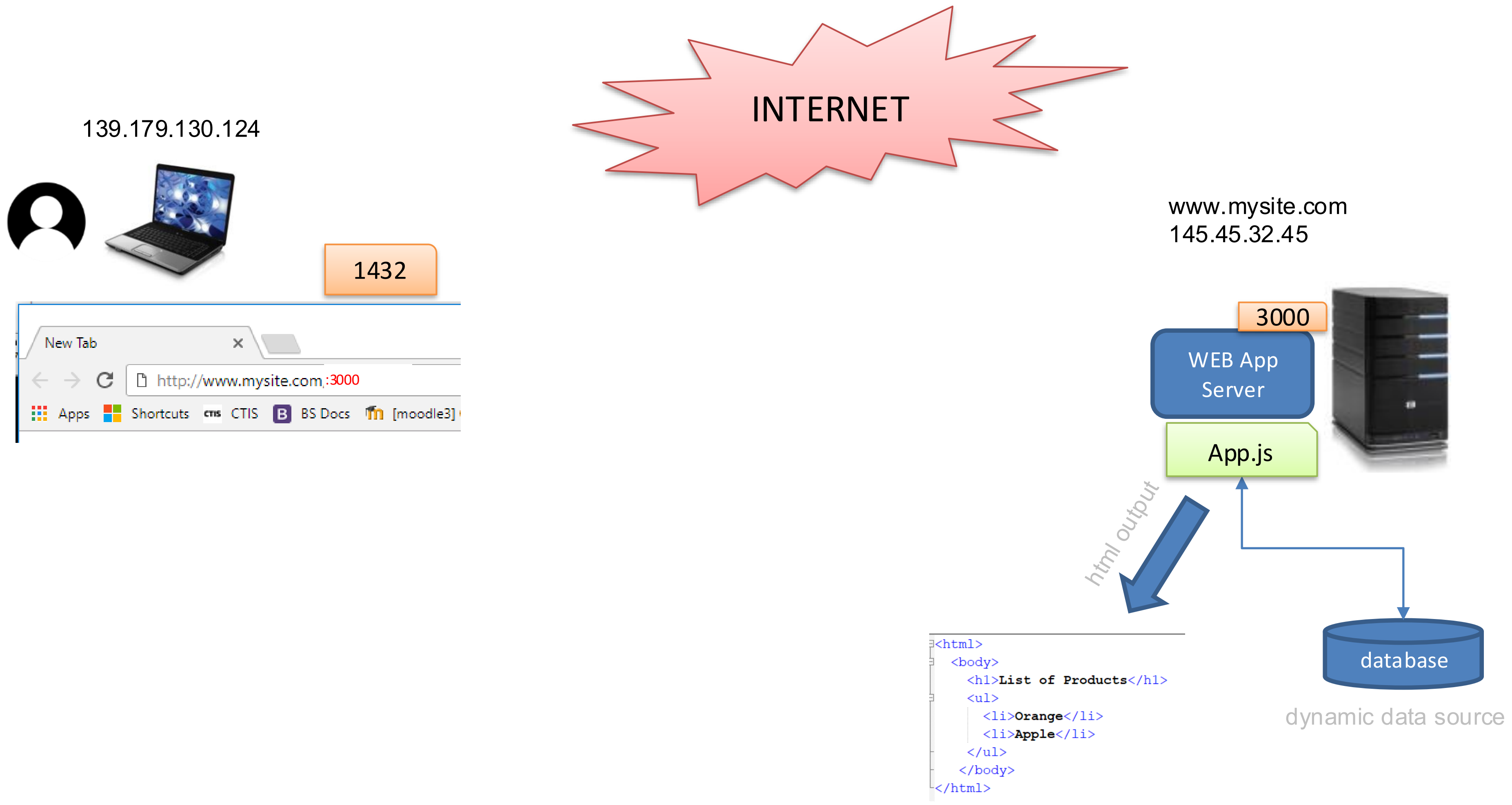






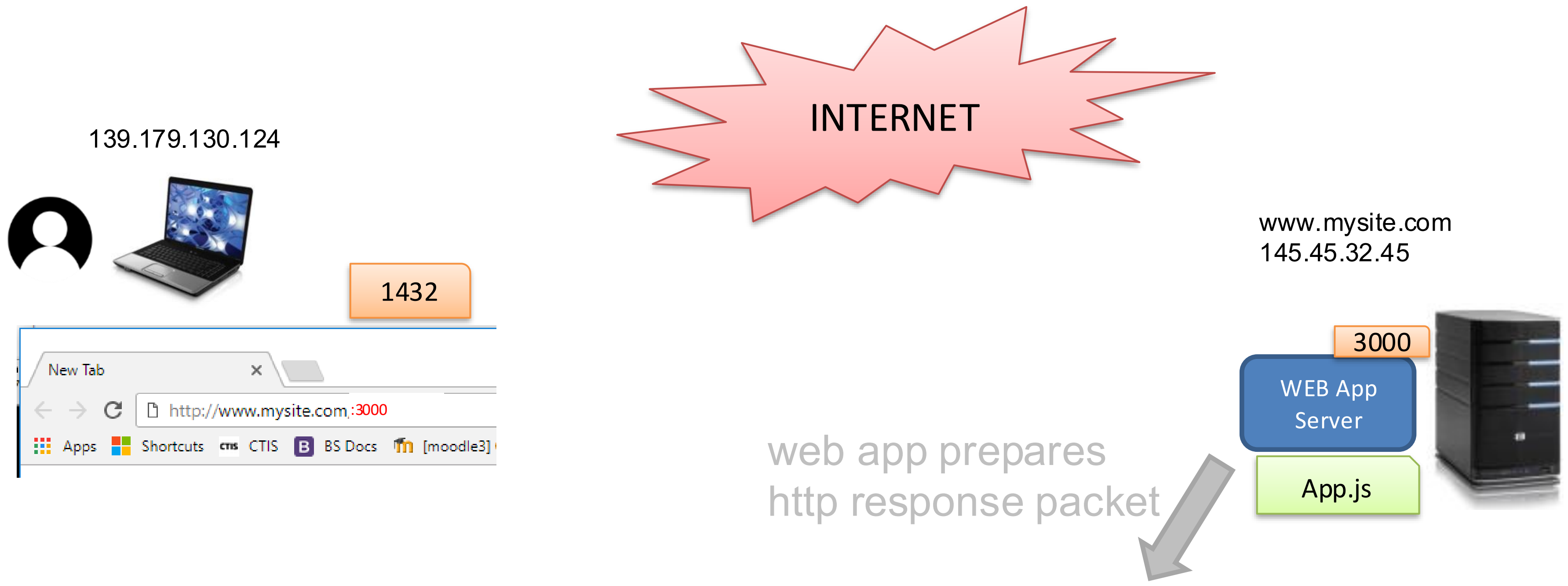
145.45.32.45 (dest IP)	3000 (dest)
139.179.130.124 (src IP)	1432 (src Port)
GET / HTTP/1.1 host: www.mysite.com	

Packet arrives at Web App Server



```
<html>
  <body>
    <h1>List of Products</h1>
    <ul>
      <li>Orange</li>
      <li>Apple</li>
    </ul>
   </body>
</html>
```

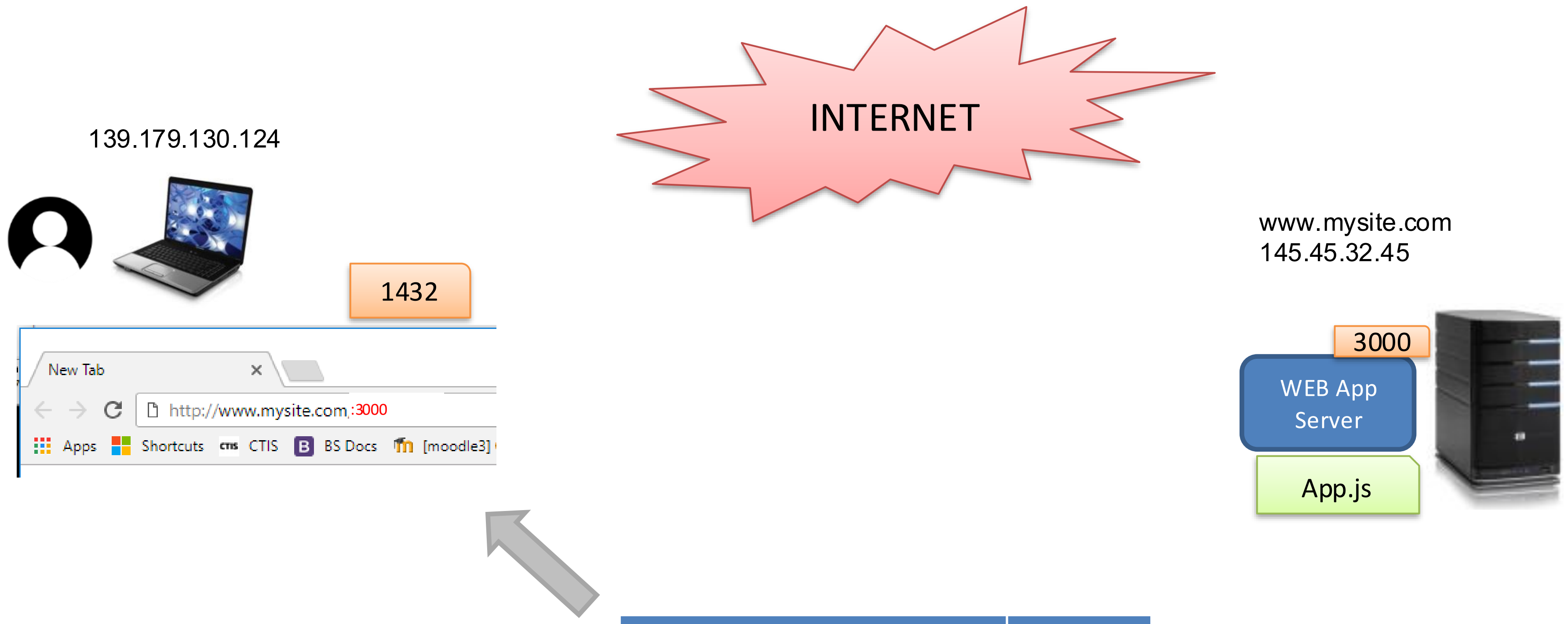
SSR (Server-Side Rendering)



139.179.130.124	1432
145.45.32.45	3000

HTTP/1.1 200 OK
Date: Sat, 15 Jan 2025 14:37:12 GMT
Server: nodejs server
Content-Type: text/html
Content-Length: 123

```
<html>
  <body>
    <h1>List of Products</h1>
    <ul>
      <li>Orange</li>
      <li>Apple</li>
    </ul>
   </body>
</html>
```

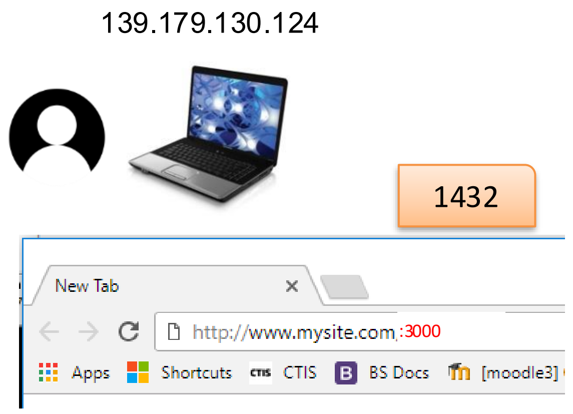


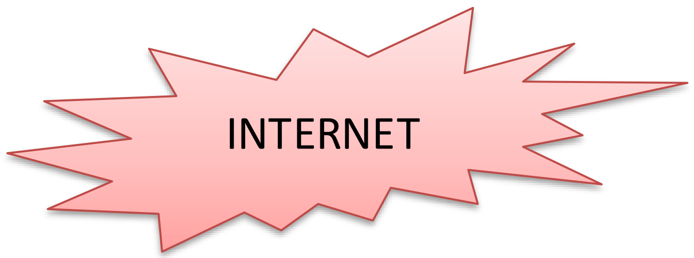
web server sends
http response packet
to web client(browser)

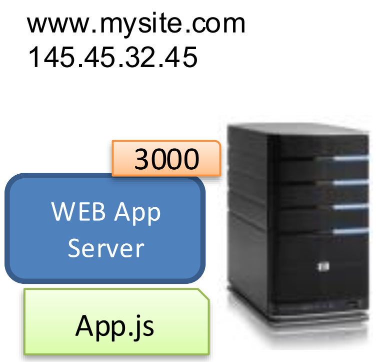
139.179.130.124	1432
145.45.32.45	3000

```
HTTP/1.1 200 OK
Date: Sat, 15 Jan 2025 14:37:12 GMT
Server: nodejs server
Content-Type: text/html
Content-Length: 123
```

```
<html>
  <body>
    <h1>List of Products</h1>
    <ul>
      <li>Orange</li>
      <li>Apple</li>
    </ul>
   </body>
</html>
```



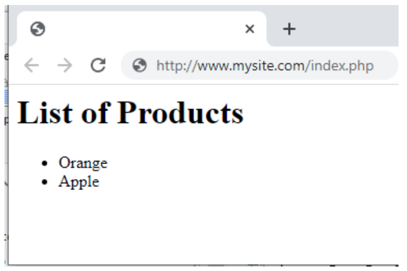




139.179.130.124	1432
145.45.32.45	3000

```
HTTP/1.1 200 OK
Date: Sat, 15 Jan 2025 14:37:12 GMT
Server: nodejs server
Content-Type: text/html
Content-Length: 123
<html>
  <body>
    <h1>List of Products</h1>
    <ul>
      <li>Orange</li>
      <li>Apple</li>
    </ul>
   </body>
</html>
```

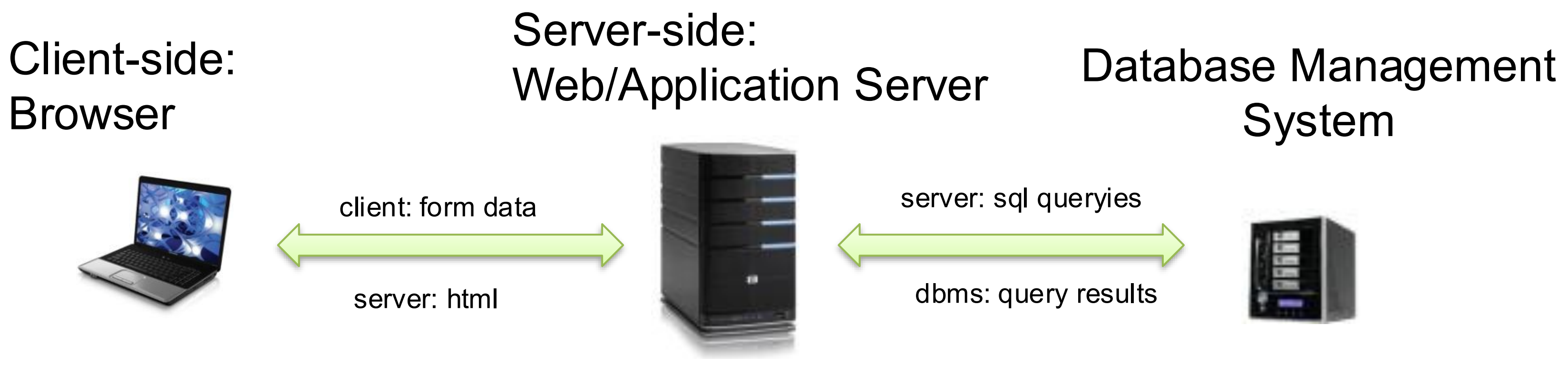
browser renders html page



List of Products

- Orange
- Apple

Three Tier Architecture



There are three basic components in Web Applications:

1. **Client-side:**
   - interface of the application (Input/Output part of Web Application)
   - displays html data produced by server-side programs (nodejs)
   - gets input from users mostly through html forms
   - send form data to server-side programs
2. **Server-side:**
   - All server-side programs(nodejs, php, etc) and resources (image, html, pdf etc) reside in server-side.
   - It gets data from client-side, processes them, and generates html codes
   - All business logic are implemented in server-side.
3. **DBMS:**
   - It may be in the same server machine with application server or may be in another machine.
   - Gets queries from server-side programs, and return results to them.

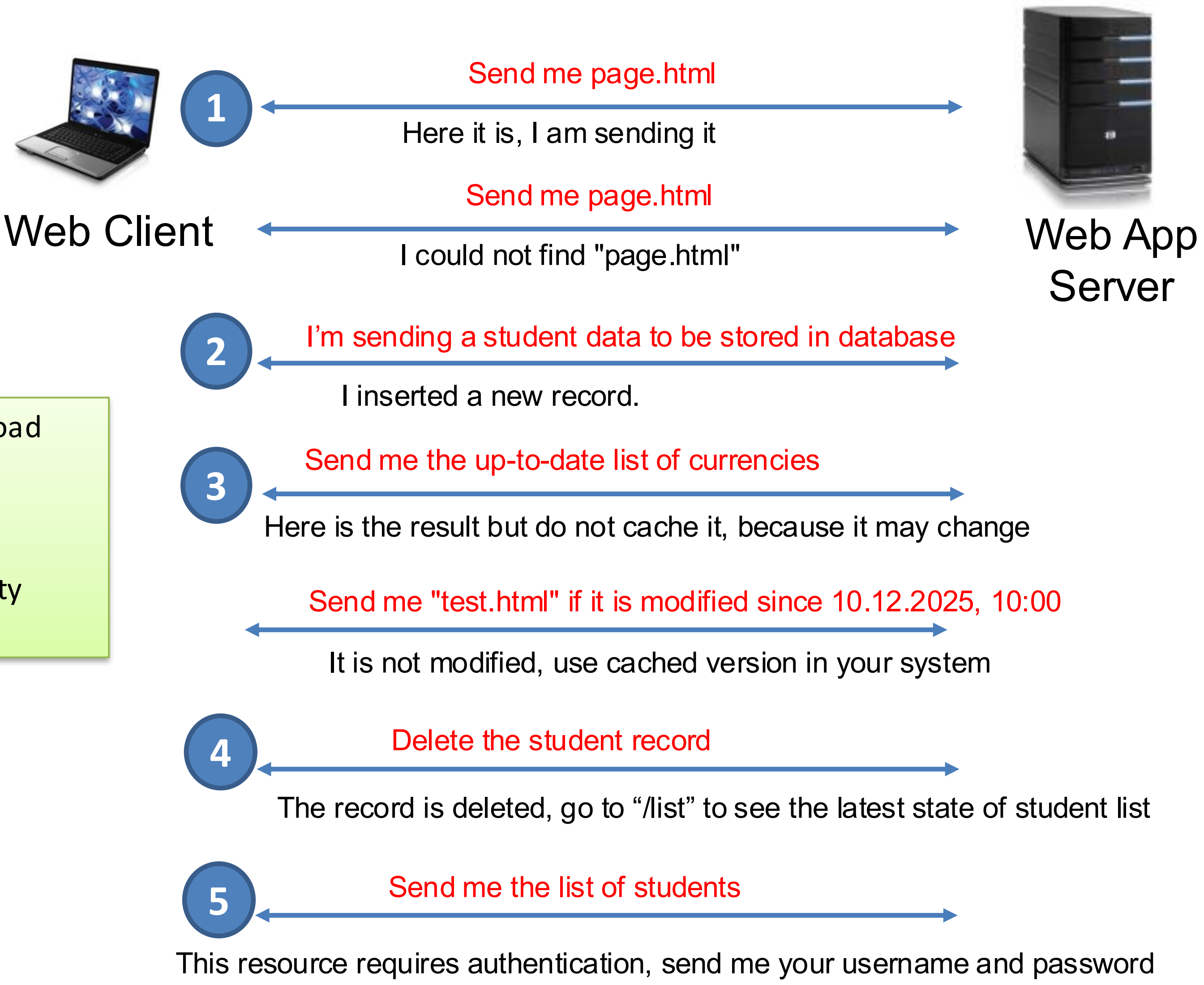
HTTP Protocol Basics

- The language between Web Clients and Web Servers.
- Basically, it defines how a client sends requests to a server and vice versa.
- Need to know HTTP protocol for redirection, cookie and session management, and cache management.
- You can use WireShark tool/Built-in Developer Tools/Browser Plugins to analyze HTTP packets between client and server.

References:

- ***HTTP/1.1:*** *"HTTP: The Definitive Guide" – David Gourley & Brian Totty, O'Reilly Media, 2002*
- ***HTTP/2:*** *"HTTP/2 in Action" – Barry Pollard, Manning, 2019*

Conversation Scenarios



1

Send me page.html

Here it is, I am sending it

Send me page.html

I could not find "page.html"

2

I’m sending a student data to be stored in database

I inserted a new record.

3

Send me the up-to-date list of currencies

Here is the result but do not cache it, because it may change

Send me "test.html" if it is modified since 10.12.2025, 10:00

It is not modified, use cached version in your system

4

Delete the student record

The record is deleted, go to “/list” to see the latest state of student list

5

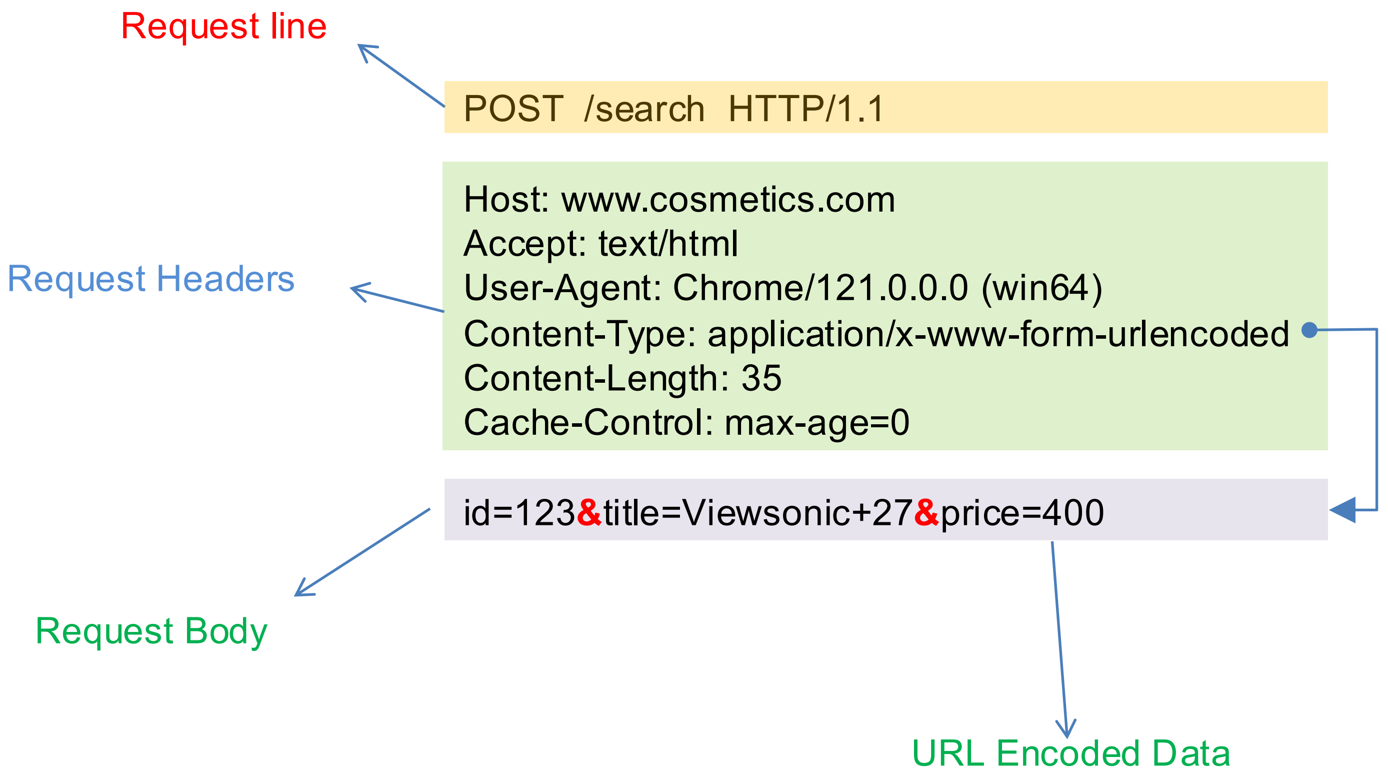
Send me the list of students

This resource requires authentication, send me your username and password

1. File Request to download
2. Run backend program
3. Caching
4. Redirection
5. Authentication/Security

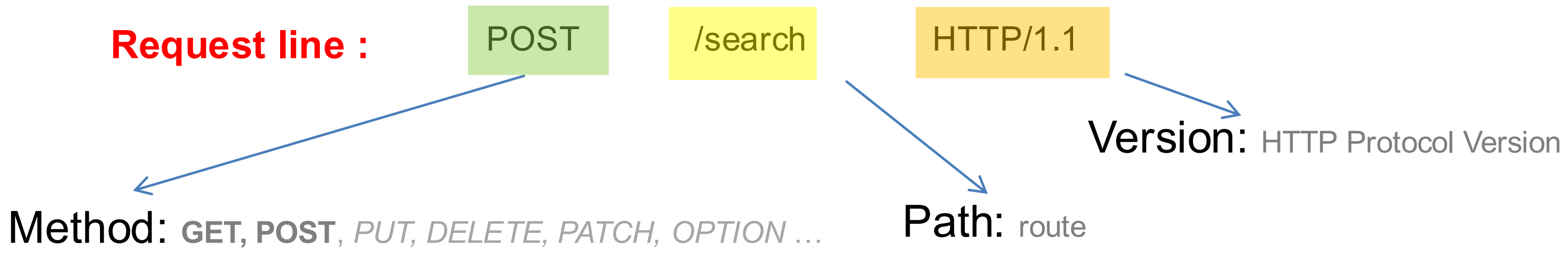
HTTP Request

- Client sends an http request packet to the server about the file/resource it needs.



```
POST  /search  HTTP/1.1

Host: www.cosmetics.com
Accept: text/html
User-Agent: Chrome/121.0.0.0 (win64)
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Cache-Control: max-age=0

id=123&title=Viewsonic+27&price=400
```

Request Line


Request line :
POST
/search
HTTP/1.1
Method: GET, POST, PUT, DELETE, PATCH, OPTION …
Path: route
Version: HTTP Protocol Version

- **GET**: Request the resource located at the specified URL
  - When you **write a URL address** to address bar in the browser, and press Enter key, the browser automatically generates GET request packet, and send it to the server.
  - When you **click on a link**, the browser generates GET request packet.
  - It is also possible to send data to server-side program with GET method without using any html form at all. In the URL, **?**var1=value1 **&** var2=value2 format is used to send data using GET method.
- **POST**: Sends data to the program located at the specified URL
  - After filling an html form, and **click on a submit button**, browser generates a POST request packet. The data you filled in the form are sent to the given server-side script inside the request body in URL encoded format.

Basic Request Headers

```
POST  /addproduct  HTTP/1.1

Host: www.cosmetics.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.2.3029
Referer: https://www.previous-page.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Cookie: sessionID=abc123; userPref=dark
Connection: keep-alive

id=123&title=Viewsonic+27&price=400
```

Host : *indicates the domain name of the server since multiple domains can be hosted on the same server and share the same IP address. This is a mandatory field in HTTP1.1.*

User-Agent : *provides information about the user agent (e.g. the browser or client) making the request. A backend program can deduce the browser type, operating system and the platform (mobile, desktop etc.) from User-Agent header.*

Accept : *informs the server about the types of media (e.g. MIME types) that the client can handle.*

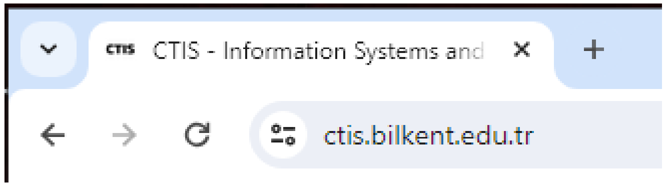
Accept-Encoding *: lists the encoding methods supported by the client for the response content.*

Referer : *specifies the URL of the page that linked to the resource being requested*

Cookie : *sends cookies previously stored on the client side to the server (will be explained later*)

Content-Type *: indicates the media type (also known as MIME type) of the resource being sent in the request body. It informs the server about the format of the data so that the server can properly interpret and process the payload.*

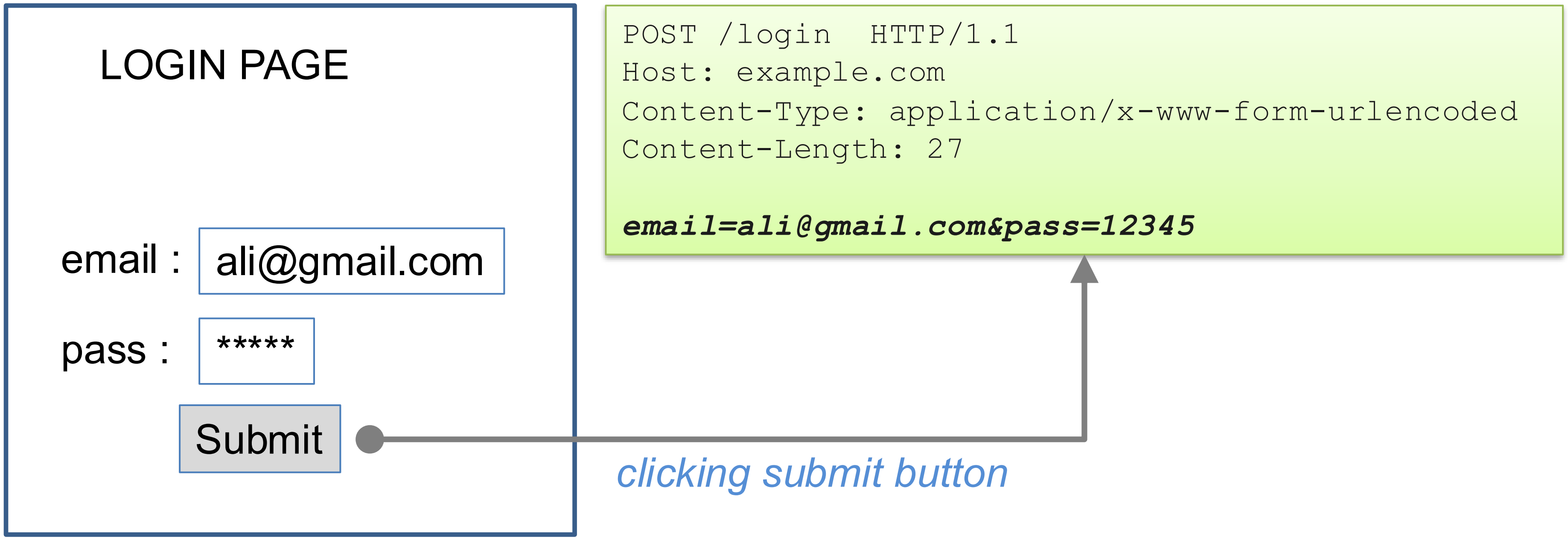
GET Request Sample



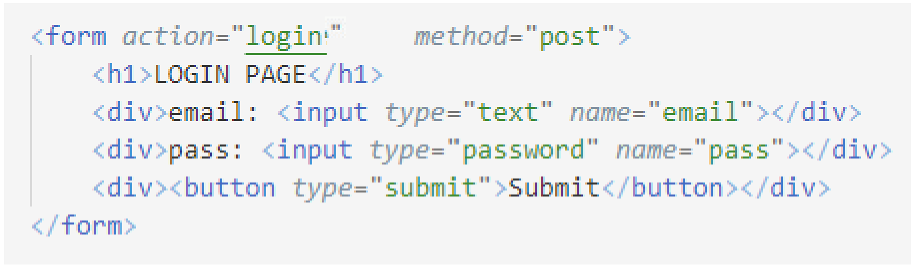
type url and press enter

```
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.
hange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,tr-TR;q=0.8,tr;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _ga=GA1.3.1107871222.1600351982; _ga_677E02R79Z=GS1.
Host: www.ctis.bilkent.edu.tr
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleW
```

POST Request Sample



```
POST /login  HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

email=ali@gmail.com&pass=12345
```



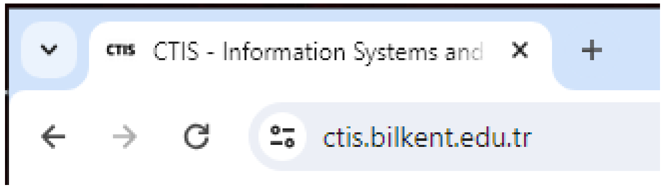
```
<form action="login"     method="post">
    <h1>LOGIN PAGE</h1>
    <div>email: <input type="text" name="email"></div>
    <div>pass: <input type="password" name="pass"></div>
    <div><button type="submit">Submit</button></div>
</form>
```

Browser automatically generates a POST request to a specified route involving form data. This is how a user can submit/upload data to a web application.

HTTP Response

- Application/Web server sends the results in HTTP response packet format.
- Format:
  - **Status line**: version, status code, description
    - 200 OK, 206 Partial Content, 404 Not Found, 301 Moved permanently
    - 302 Found, 401 Unauthorized, 403 Forbidden
    - 500 Internal Server Error
  - **Response Headers**
    - Date, Content-Type, Content-Length, Location, Server, Content-Encoding
    - Set-Cookie, WWW-Authenticate, Cache-Control, ETag, Last-Modified
  - **Response Body**: contains the payload.

HTTP Response Example



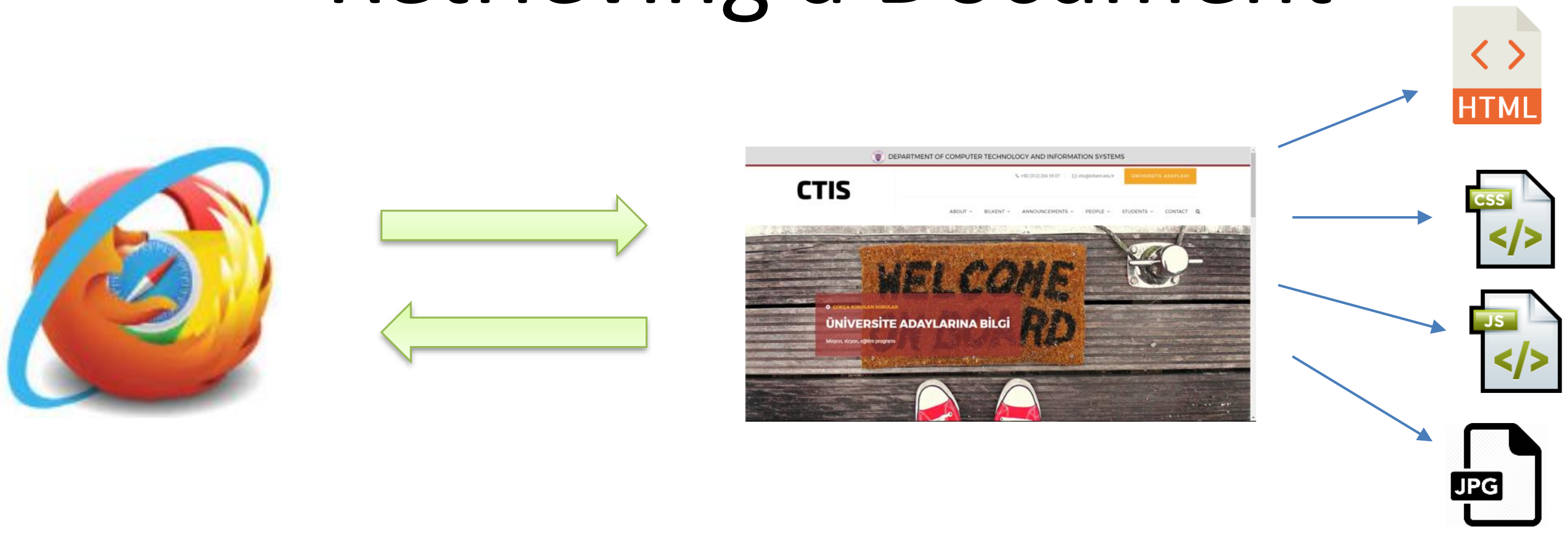
```
HTTP/1.1 200 OK
Date: Thu, 01 Feb 2024 10:21:14 GMT
Server: Apache/2.4.56 (Debian)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 13872
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<!--[if IE 9]>
<html class="ie ie9" lang="en-US">
<![endif]-->
<html lang="en-US">
<head>
```

Status Line

HTTP Response Headers

Response Body

Retrieving a Document


DEPARTMENT OF COMPUTER TECHNOLOGY AND INFORMATION SYSTEMS
CTIS
ÜNİVERSİTE ADAYLARINA BİLGİ
HTML
CSS
JS
JPG

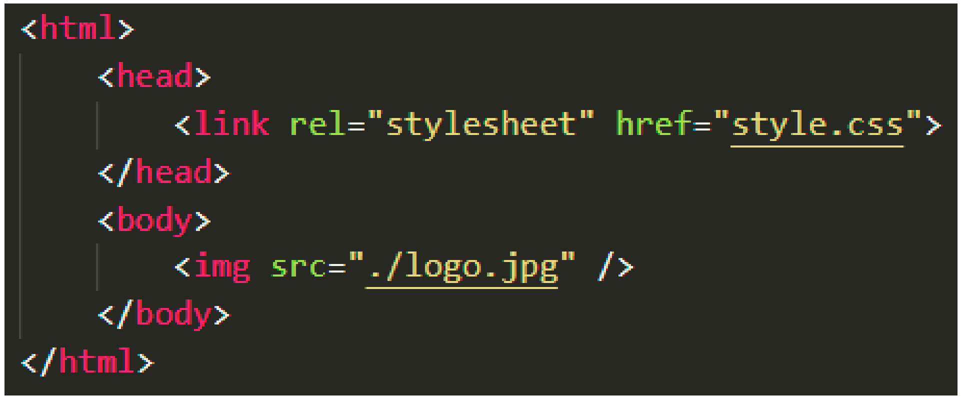
Rendering a document:

- A document includes several files such as an HTML file, along with one or more CSS, JavaScript, image, and font files.
- When you request a page, the browser downloads all necessary files (dependencies) one by one to be able to render the document.
- It sends http requests for each file to download.

Example

Request URL: http://one.net/index.html

index.html *(origin resource)*



```
<html>
    <head>
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <img src="./logo.jpg" />
    </body>
</html>
```

1. GET index.html

The browser parses the html codes and finds out all dependencies (***style.css***, ***logo.jpg***), and downloads all of them with http, one by one.

2. GET style.css

style.css

3. GET logo.jpg

logo.jpg

After downloading all dependencies, it is ready to render the page.

4. Now, It can render the document (index.html)

curl

- curl (Client URL) is a web client, command line tool used to send http requests.
- curl is widely used for downloading/uploading and testing Web APIs.
- curl easily integrates into scripts and automation workflows.
- It is already available on all major operating systems.

Basic Usage Examples:

1. Sending GET Request *(-v : verbose mode)*

```
C:\>curl http://www.one.net -v
```

2. Sending POST Request *(default Content-Type: x-www-urlencoded format, -d: add post data, -X request method (default is GET))*

```
C:\>curl -X POST -d "name=john&age=30" http://www.one.net/Person
```

3. Sending POST Request *(payload: application/json) (-H: add a request header)*

```
C:\>curl -X POST -H "Content-Type: application/json"
   -d "{\"name\": \"john\", \"age\": \"30\"}"
   http://www.one.net/Person
```

(will be used later)

4. Download a file

```
C:\>curl -O https://example.com/file.zip
```

OPTIONAL

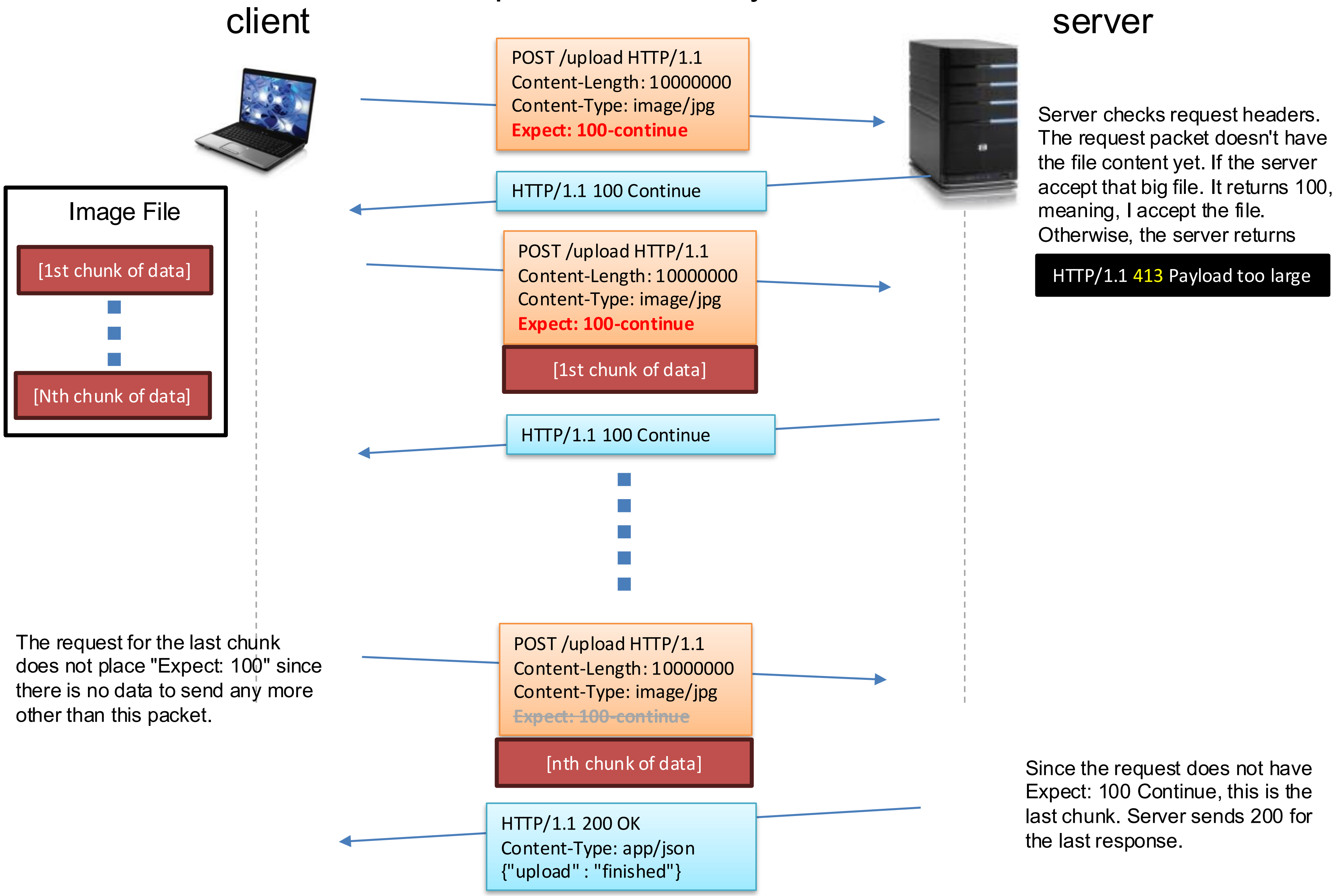
(Not Included in the Course)

HTTP Status Codes

- 100 – Continue: upload big data
- 101 – Switching Protocols
- 200 – OK
- 201 – Created
- 202 – Accepted
- 203 – Non-Authoritative Information
- 204 – No Content
- 205 – Reset Content
- 206 – Partial Content

100 - Continue

- To upload big amount of file (data)
- Prevents Unnecessary Data Transfer
- Improves Efficiency



```
POST /upload HTTP/1.1
Content-Length: 10000000
Content-Type: image/jpg
Expect: 100-continue
```

```
HTTP/1.1 100 Continue
```

Server checks request headers. The request packet doesn't have the file content yet. If the server accept that big file. It returns 100, meaning, I accept the file. Otherwise, the server returns

```
HTTP/1.1 413 Payload too large
```

```
POST /upload HTTP/1.1
Content-Length: 10000000
Content-Type: image/jpg
Expect: 100-continue
[1st chunk of data]
```

```
HTTP/1.1 100 Continue
```

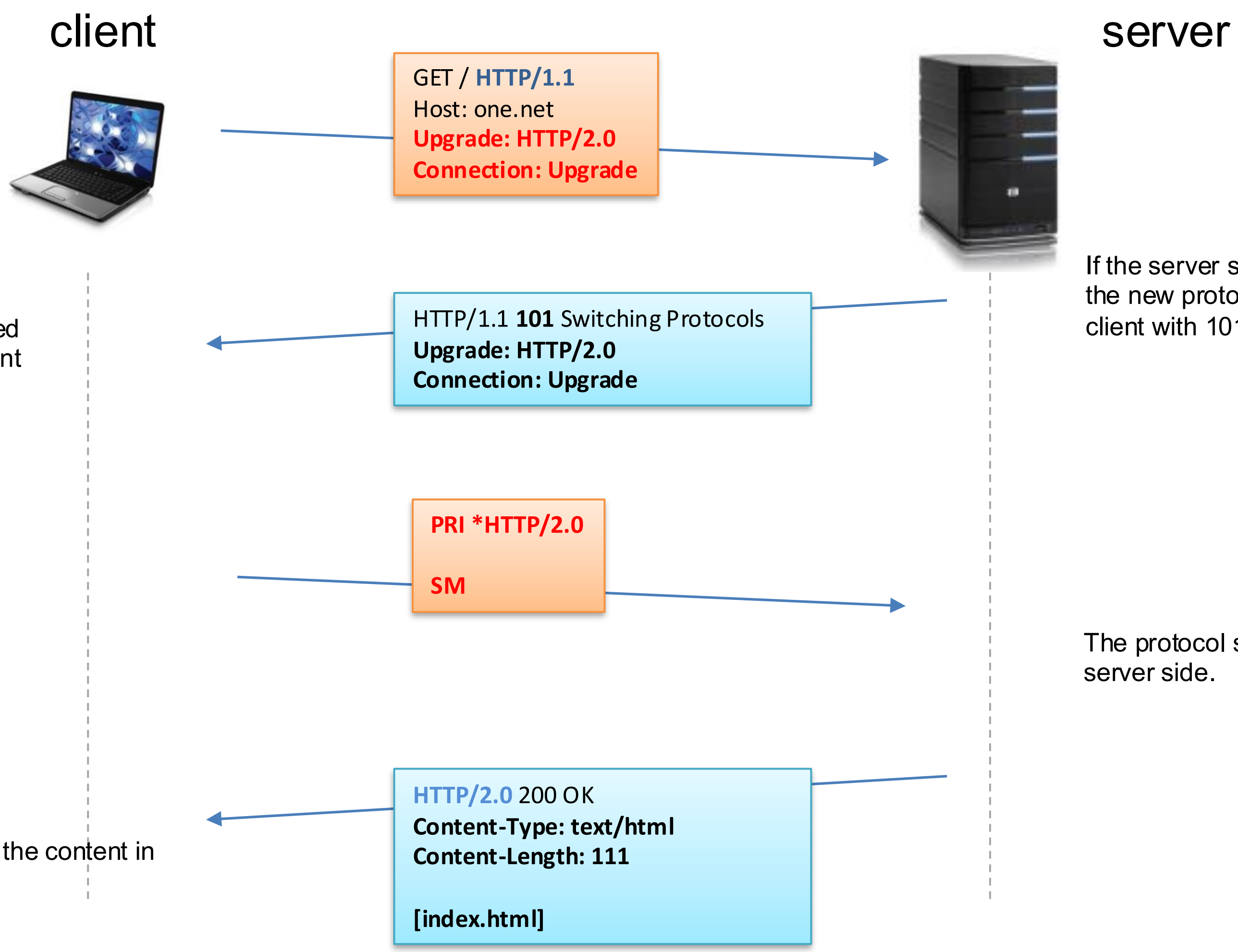
The request for the last chunk does not place "Expect: 100" since there is no data to send any more other than this packet.

```
POST /upload HTTP/1.1
Content-Length: 10000000
Content-Type: image/jpg
~~Expect: 100-continue~~
[nth chunk of data]
```

```
HTTP/1.1 200 OK
Content-Type: app/json
{"upload" : "finished"}
```

Since the request does not have Expect: 100 Continue, this is the last chunk. Server sends 200 for the last response.

101 – Switching Protocols



```
GET / HTTP/1.1
Host: one.net
Upgrade: HTTP/2.0
Connection: Upgrade
```

If the server supports and agrees the new protocol, It notifies the client with 101 status code.

```
HTTP/1.1 101 Switching Protocols
Upgrade: HTTP/2.0
Connection: Upgrade
```

The server accepted HTTP/2.0. The client sends Protocol Required Indicator (**PRI**) message to server.

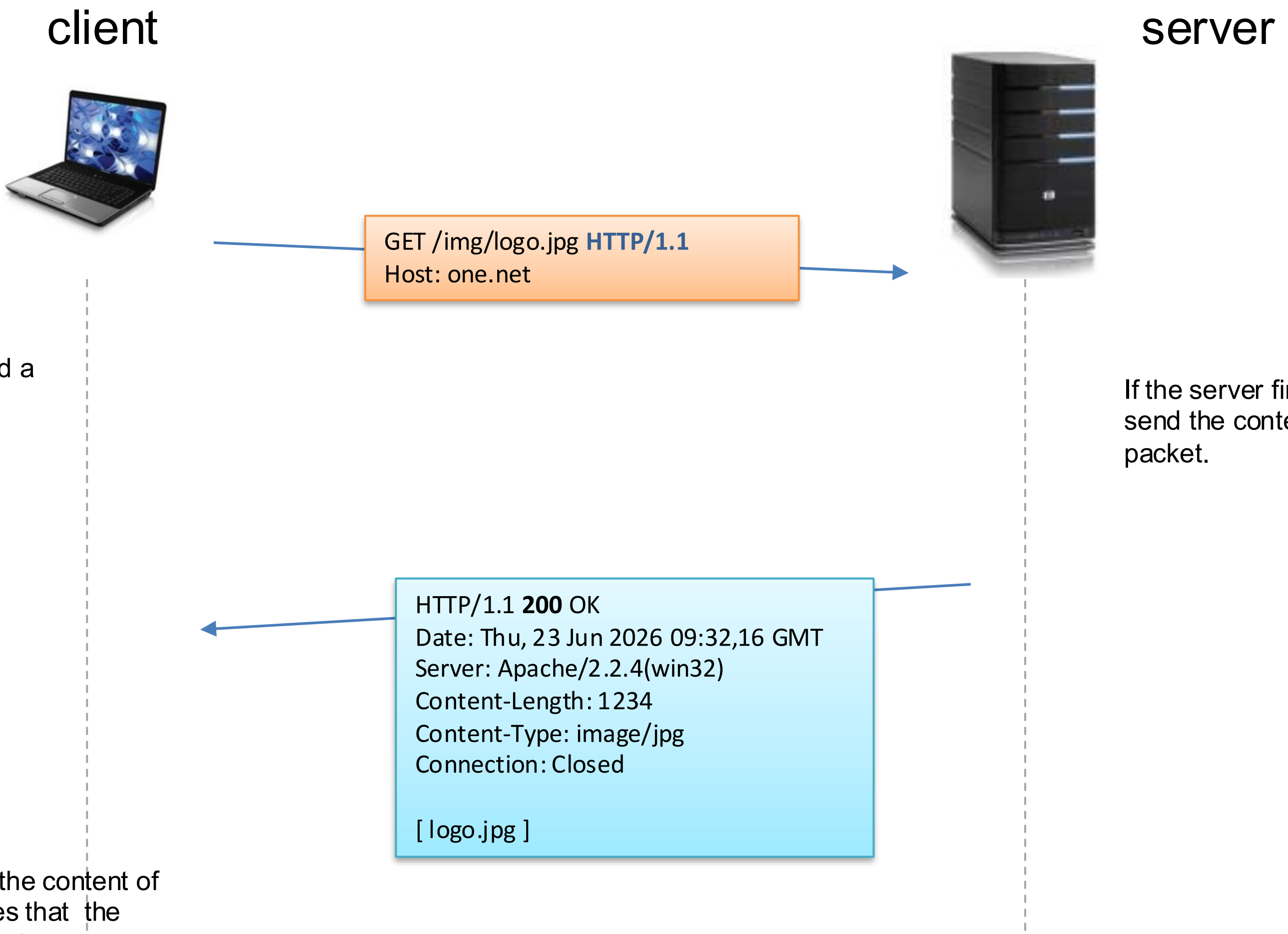
```
PRI *HTTP/2.0

SM
```

The protocol switch occurs on server side.

```
HTTP/2.0 200 OK
Content-Type: text/html
Content-Length: 111

[index.html]
```

The client receives the content in HTTP/2.0 protocol.

200 – OK



```
GET /img/logo.jpg HTTP/1.1
Host: one.net
```

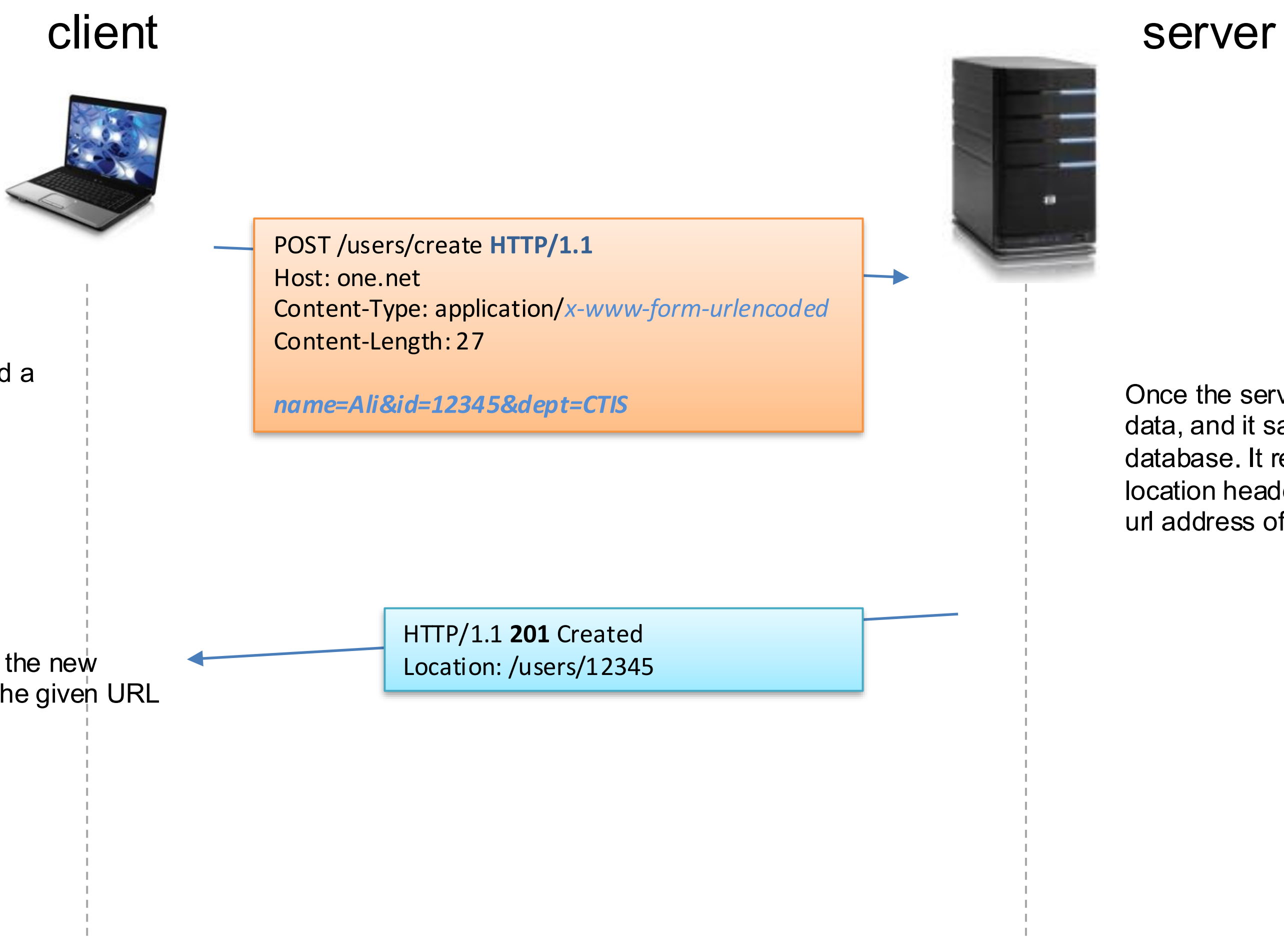
The client sends a request to download a resource.

If the server finds this resourse, it send the content in response packet.

```
HTTP/1.1 200 OK
Date: Thu, 23 Jun 2026 09:32,16 GMT
Server: Apache/2.2.4(win32)
Content-Length: 1234
Content-Type: image/jpg
Connection: Closed

[ logo.jpg ]
```

The client recieves the content of the file. 200 indicates that the request was successful.

201 – Created



The client sends a request to download a resource.

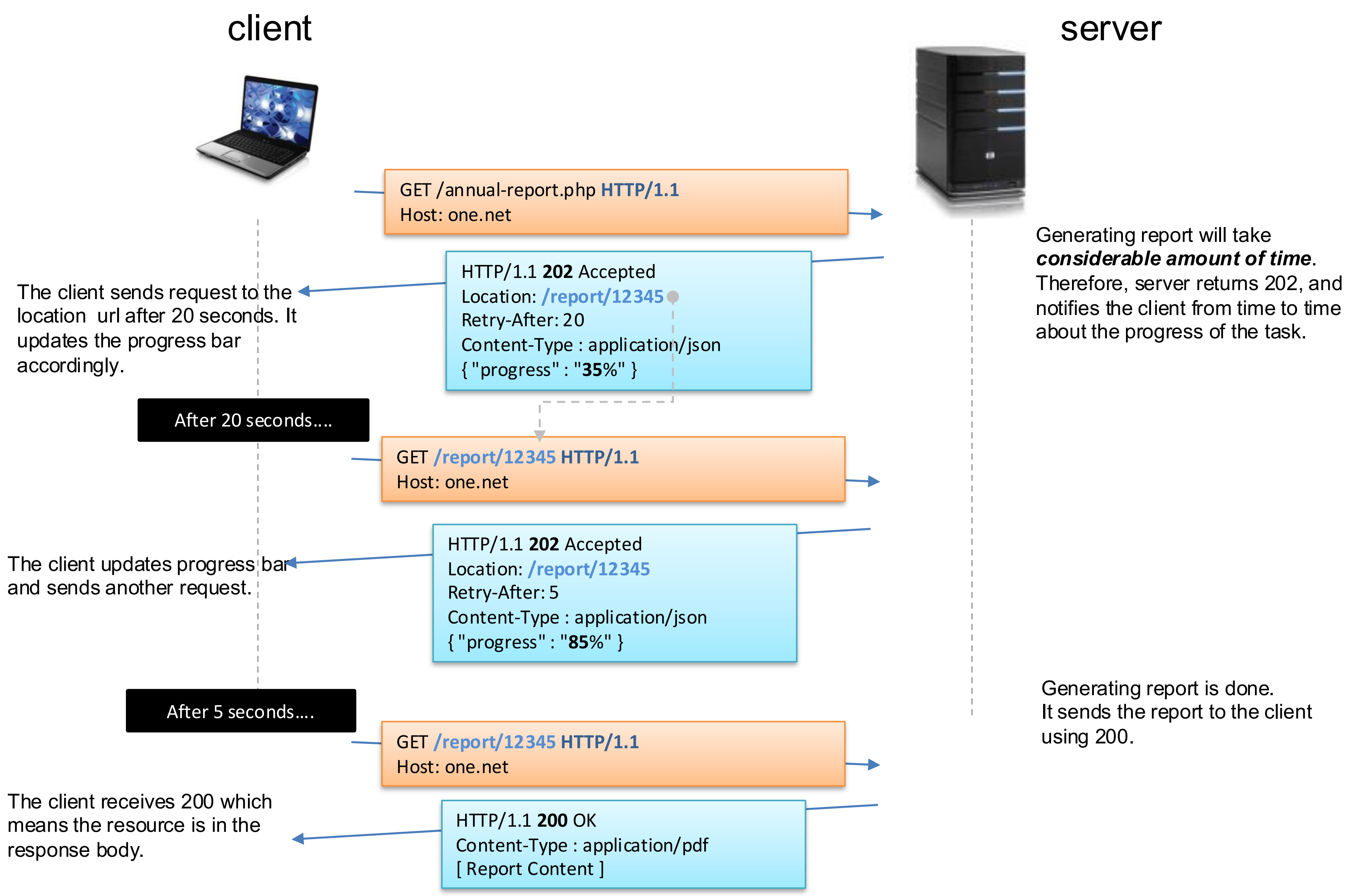
```
POST /users/create HTTP/1.1
Host: one.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

name=Ali&id=12345&dept=CTIS
```

Once the server receives the form data, and it saves them into the database. It returns 201, and the location header that indicates the url address of the new resource.

```
HTTP/1.1 201 Created
Location: /users/12345
```

The client can show the new record by following the given URL address.

202 – Accepted



```
GET /annual-report.php HTTP/1.1
Host: one.net
```

Generating report will take ***considerable amount of time***. Therefore, server returns 202, and notifies the client from time to time about the progress of the task.

```
HTTP/1.1 202 Accepted
Location: /report/12345
Retry-After: 20
Content-Type : application/json
{ "progress" : "35%" }
```

The client sends request to the location url after 20 seconds. It updates the progress bar accordingly.

```
GET /report/12345 HTTP/1.1
Host: one.net
```

```
HTTP/1.1 202 Accepted
Location: /report/12345
Retry-After: 5
Content-Type : application/json
{ "progress" : "85%" }
```

The client updates progress bar and sends another request.

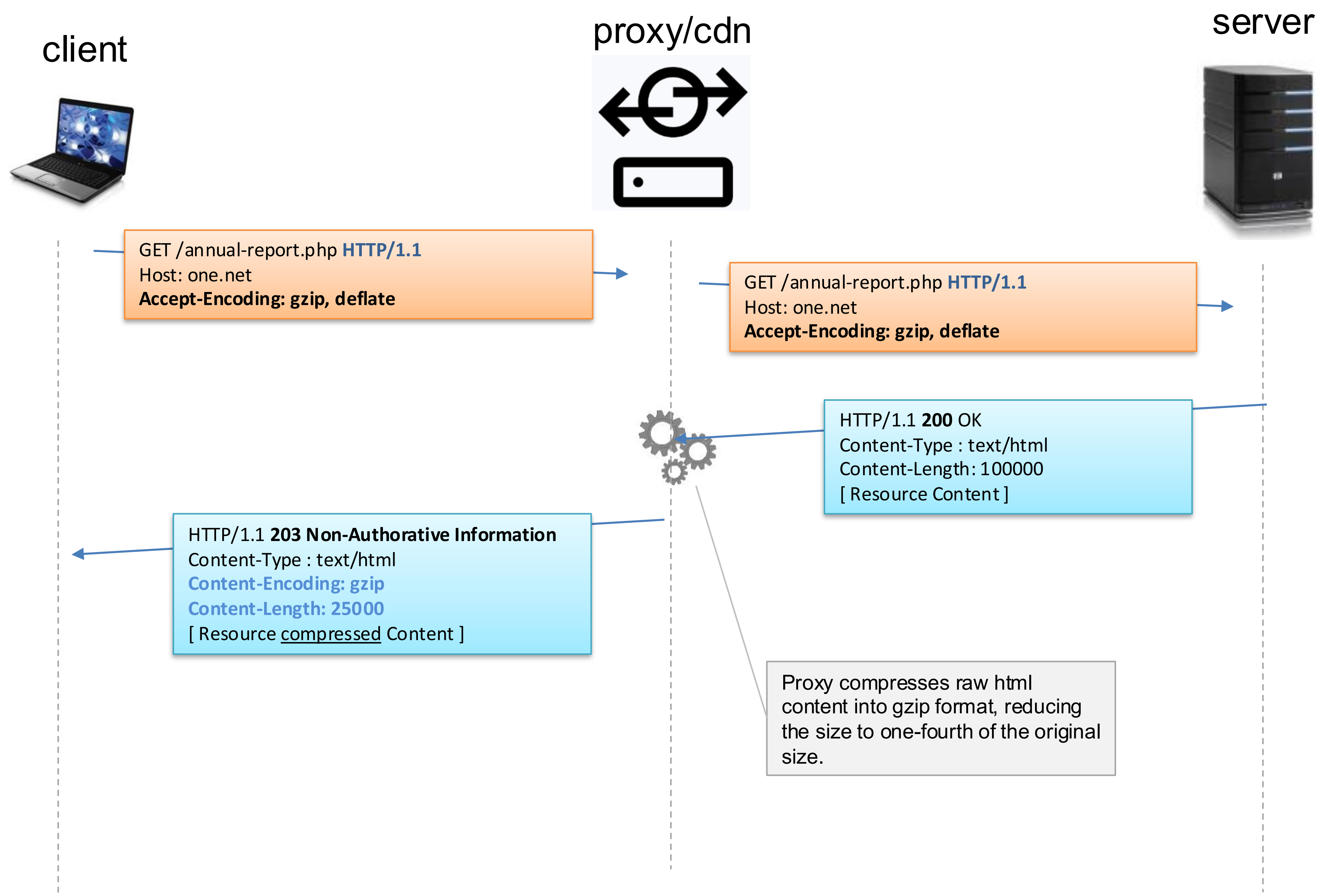
Generating report is done.
It sends the report to the client using 200.

```
GET /report/12345 HTTP/1.1
Host: one.net
```

```
HTTP/1.1 200 OK
Content-Type : application/pdf
[ Report Content ]
```

The client receives 200 which means the resource is in the response body.

203 – Non-Authorative Information



GET /annual-report.php **HTTP/1.1**
Host: one.net
Accept-Encoding: gzip, deflate

GET /annual-report.php **HTTP/1.1**
Host: one.net
Accept-Encoding: gzip, deflate

HTTP/1.1 **200** OK
Content-Type : text/html
Content-Length: 100000
[Resource Content]

HTTP/1.1 **203 Non-Authorative Information**
Content-Type : text/html
Content-Encoding: gzip
Content-Length: 25000
[Resource compressed Content]

Proxy compresses raw html content into gzip format, reducing the size to one-fourth of the original size.

204 – No Content

Used for HEAD, PUT, DELETE requests

client

server

The client wants to gather information about logo.jpg. It doesn't need the content.

```
HEAD /logo.jpg HTTP/1.1
Host: one.net
```

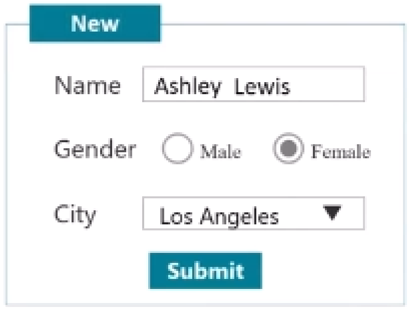
The server returns only the response headers, not the content.

```
HTTP/1.1 204 No Content
Server: Apache/2.5.7 (win32)
Last-Modified: Sun, 23 Jan 2027 15:23:12 GMT
Content-Lenght: 1234
Content-Type: image/jpeg
```

The client receives the information about the file.

205 – Reset Content

client



server

The user fills out the form and submits it.

```
POST /user/create HTTP/1.1
Host: one.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

name=Ashley+Lewis&gender=F&city=Los+Angeles
```

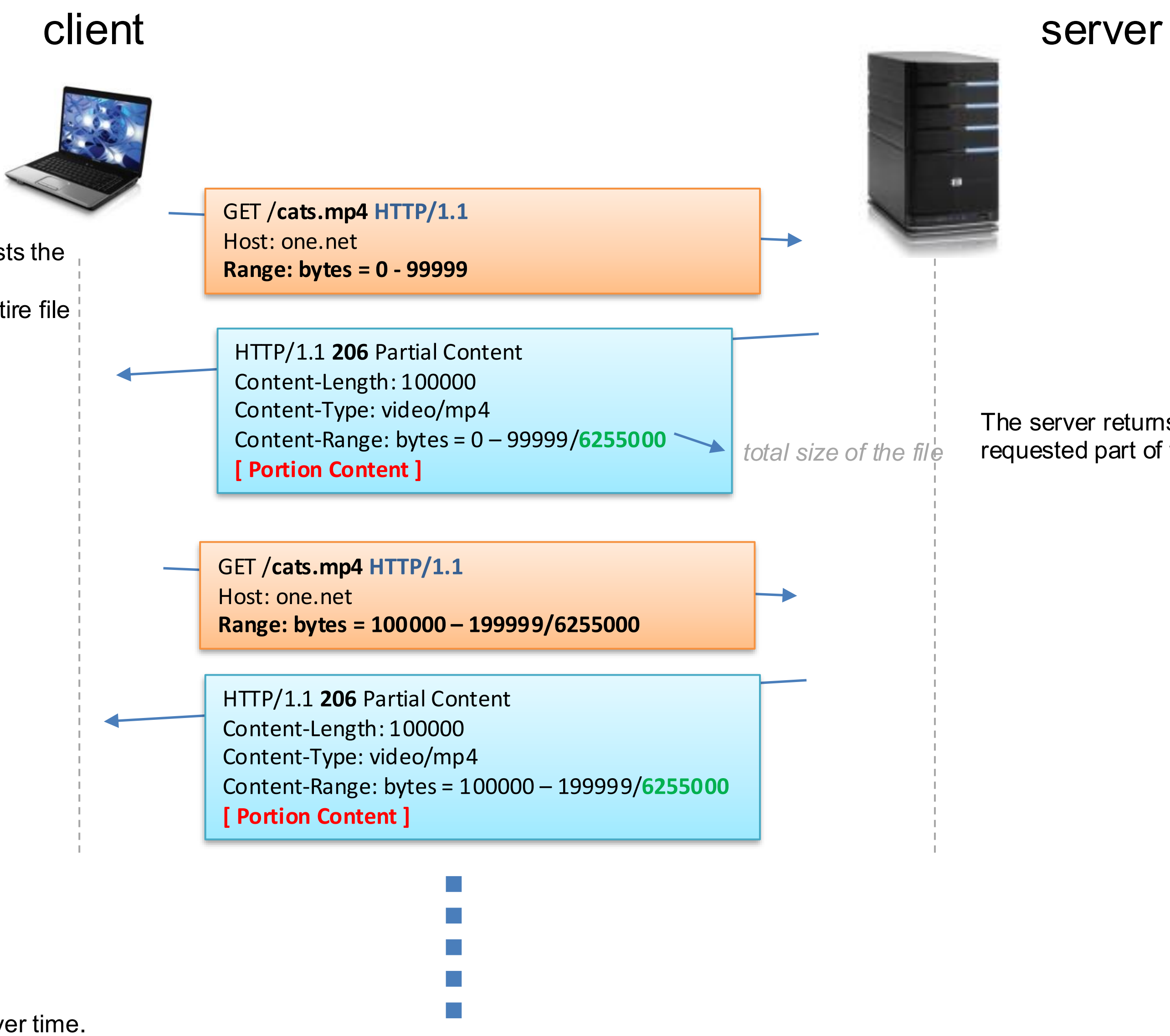
The server creates a new user. However, it wants to preserve the previous content of the web form.

```
HTTP/1.1 205 Reset Content
```

The browser does not reset the form.

206 – Partial Content

- Reduces network usage (Network Latency)
- Enhance download speeds (User Experience)



```
GET /cats.mp4 HTTP/1.1
Host: one.net
Range: bytes = 0 - 99999
```

```
HTTP/1.1 206 Partial Content
Content-Length: 100000
Content-Type: video/mp4
Content-Range: bytes = 0 – 99999/6255000
[ Portion Content ]
```

```
GET /cats.mp4 HTTP/1.1
Host: one.net
Range: bytes = 100000 – 199999/6255000
```

```
HTTP/1.1 206 Partial Content
Content-Length: 100000
Content-Type: video/mp4
Content-Range: bytes = 100000 – 199999/6255000
[ Portion Content ]
```

The client requests the resource in parts instead of the entire file at once.

The server returns only the requested part of the resource.

It collects all parts over time.

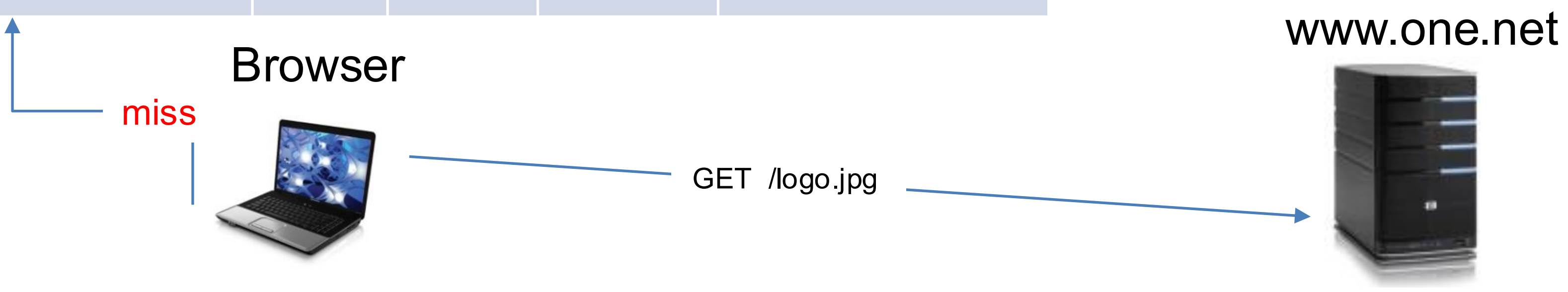
HTTP Caching

- is a mechanism to store copies of web resources (html, css, images, dynamic contents, etc) to reduce
  - server load
  - bandwidth usage
  - latency
- Key Concepts
  - **Cache** : A temporary storage area where copies of web resources are kept
  - **Cache-Control**: An HTTP header that defines a caching policies
    - **max-age**: defines the maximum time the resource is considered fresh. (*can be stale before max-age expires*)
    - **no-cache**: forces validation with the server before serving the cached copy (always validates)
    - **no-store**: prevents the resource from being stored in any cache. (always stale)
    - **private**: store only in browser cache
    - **public**: any kind of caches (browser cache, proxiy cache, etc)
  - **Expires**: An HTTP header that specifies the date/time after the response is considered stale.
  - **ETag**: An HTTP header for identifying resouce version (hash of content, or version)
  - **Last-Modified** : an HTTP header that indicates the last time the resouces was changed
    - *File Edits*: to reflect the last time the actual content of the file (HTML, CSS, JS, image,etc) was changed/edited.
    - *Database Records*: generated by backend program may place the last update time of the db entry.
    - *Dynamic Content*: it might reflect the last time the underlying data or script that generates the content.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file

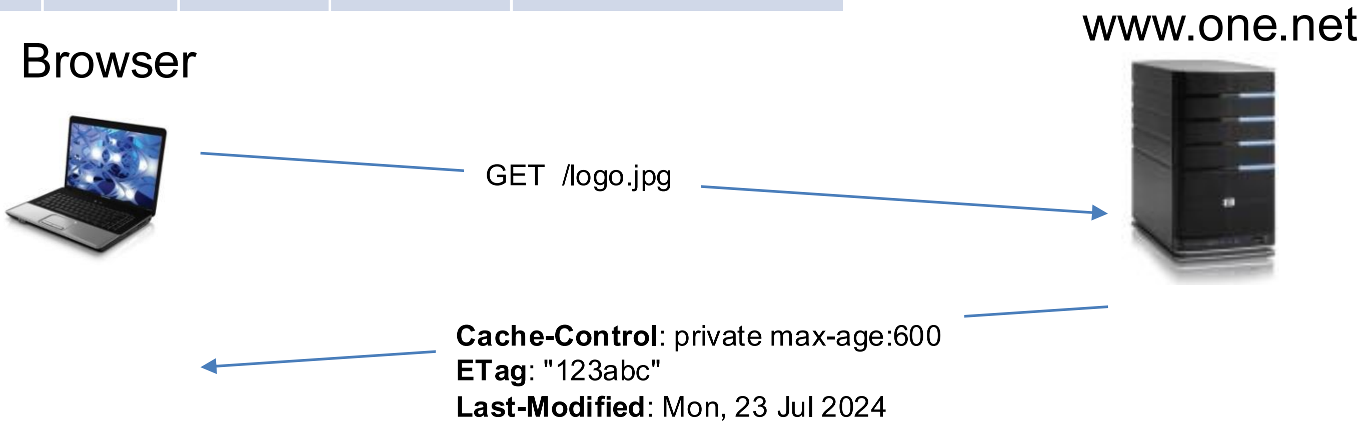


Since the cache is empty, the browser downloads logo.jpg from server the first time.
This is called "a miss".

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file



GET /logo.jpg

Cache-Control: private max-age:600
ETag: "123abc"
Last-Modified: Mon, 23 Jul 2024

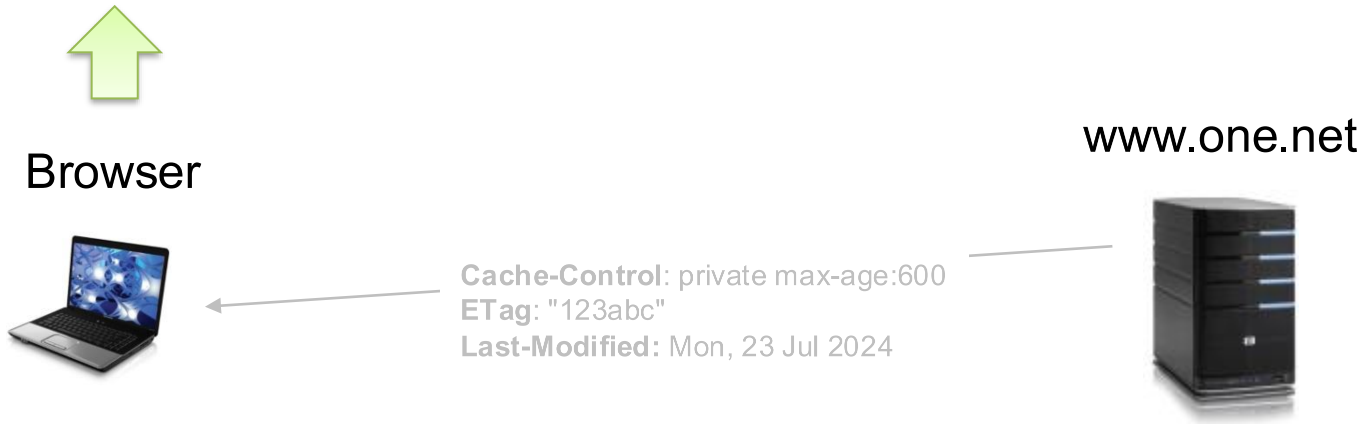
The Backend (a Web server or a backend program) sends directives about caching.
private: cache the response body on the browser only. (not in proxies)
max-age: keep the resourse for 600 seconds or 1 hour as *fresh copy*.
Etag: an identifier for the content (hash of the content, or version, or any method) to find out if the resource has been changed.
Last-Modified: shows the last timestamp the resource is updated.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	600	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp

fresh copy



Cache-Control: private max-age:600
ETag: "123abc"
Last-Modified: Mon, 23 Jul 2024

The browser inserts a new entry into the cache for "logo.jpg"
It will keep it for 600 seconds.
Etag is used for validation when it becomes stale (invalid).
Last-Modified date of the resouce is given.
"logo.jpg" is saved in a cache file of the browser.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	590	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp

hit

Browser

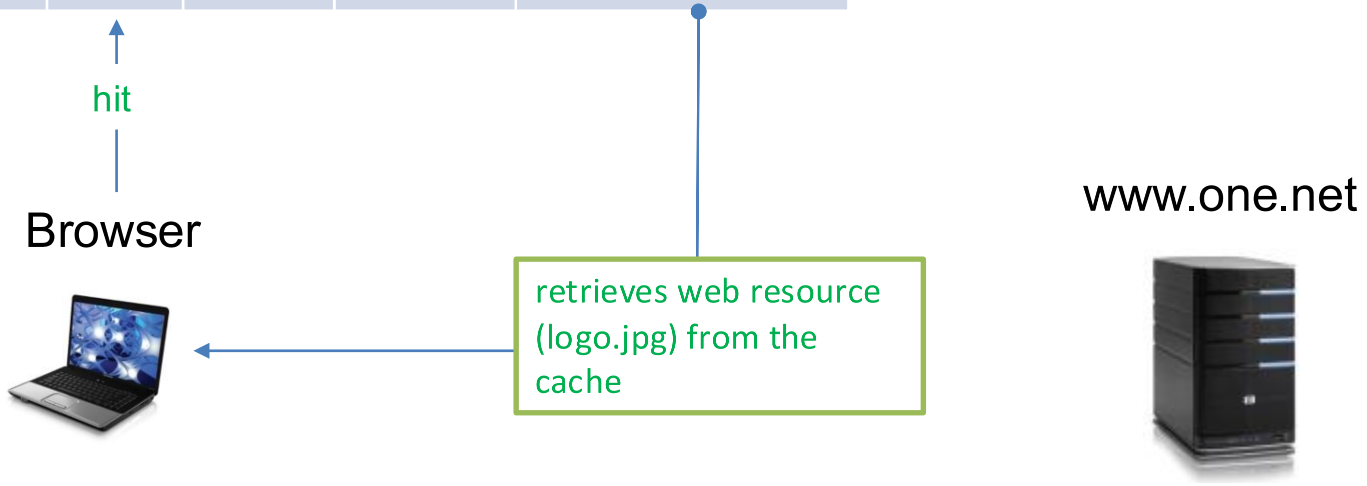
www.one.net

After 10 seconds, let the browser access logo.jpg again.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	590	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp



After 10 seconds, let the browser access logo.jpg again.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	0	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp

miss

Browser

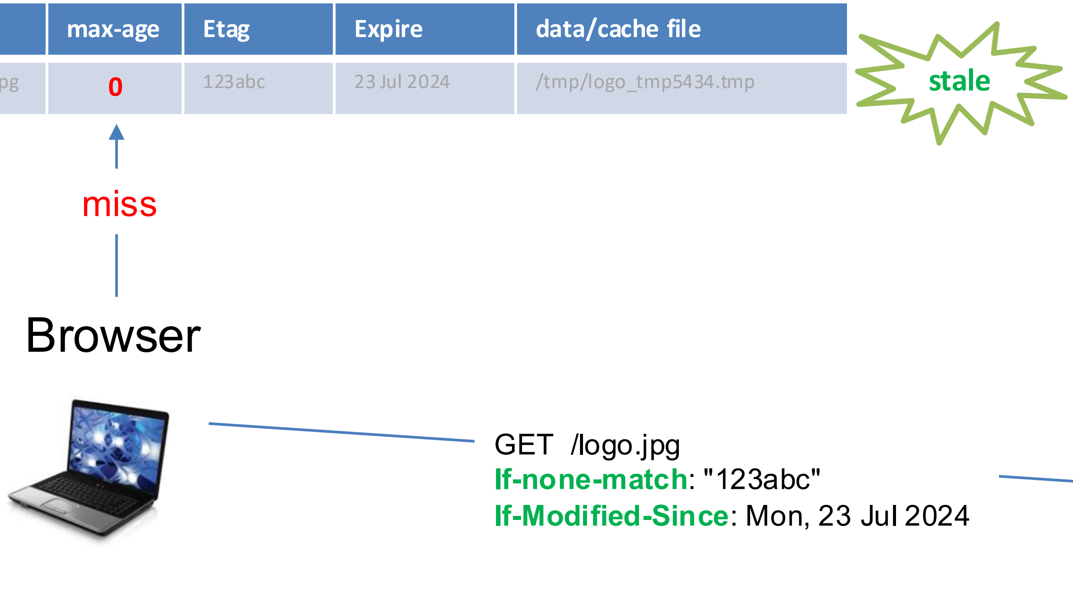
www.one.net

After 600 seconds, let the browser access logo.jpg again.
Since max-age is 0, the entry is stale now.
It is a miss to the cache.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	0	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp



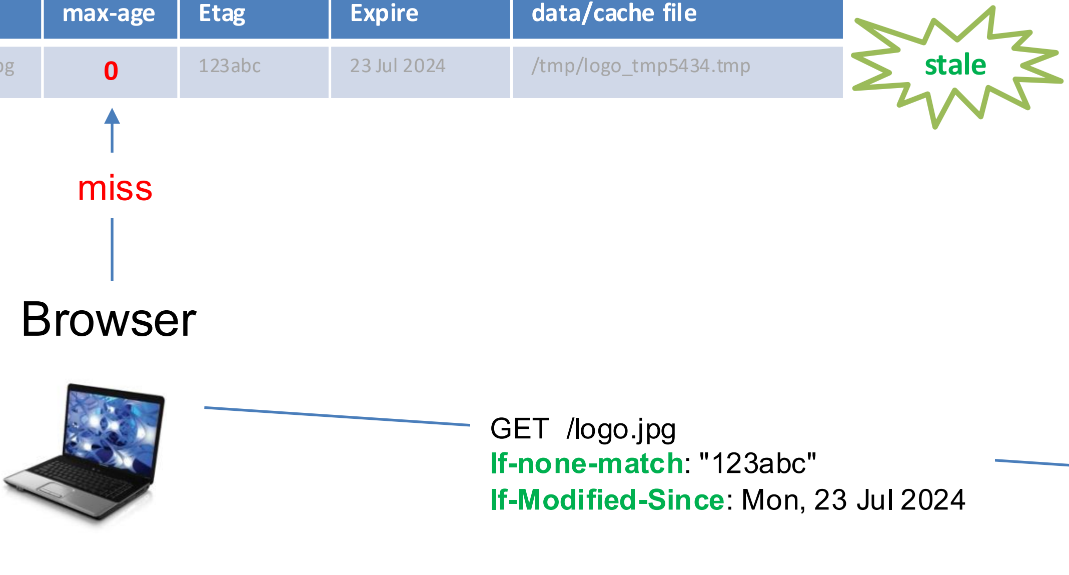
```
GET  /logo.jpg
If-none-match: "123abc"
If-Modified-Since: Mon, 23 Jul 2024
```

It needs to revalidate the web resource from the server side.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	0	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp



```
GET  /logo.jpg
If-none-match: "123abc"
If-Modified-Since: Mon, 23 Jul 2024
```

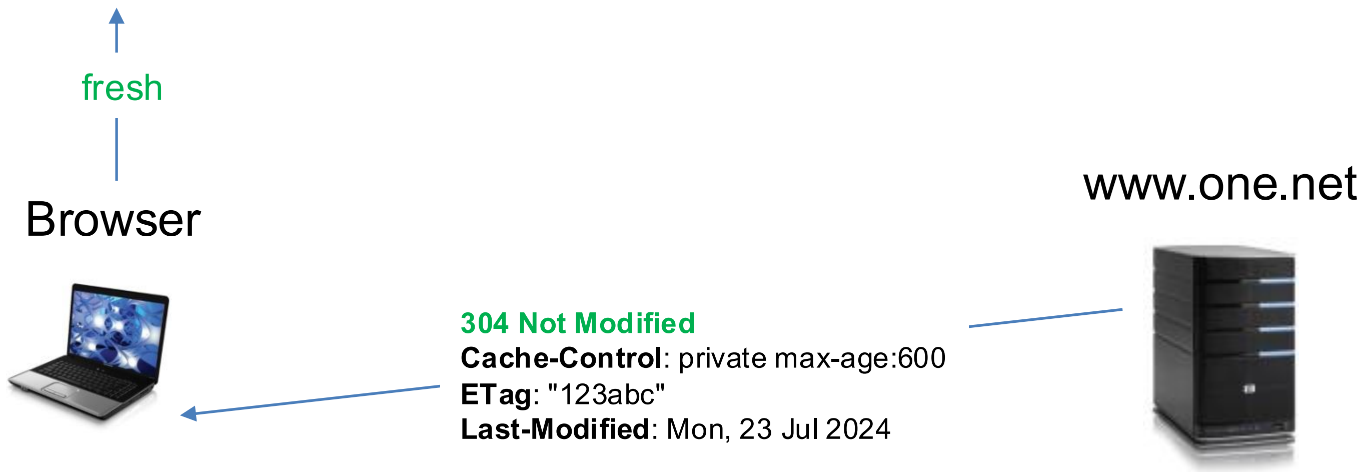
There are two options here
1. the resourse is changed or edited
2. the resourse is the same

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	600	123abc	23 Jul 2024	/tmp/logo_tmp5434.tmp

fresh copy



304 Not Modified
Cache-Control: private max-age:600
ETag: "123abc"
Last-Modified: Mon, 23 Jul 2024

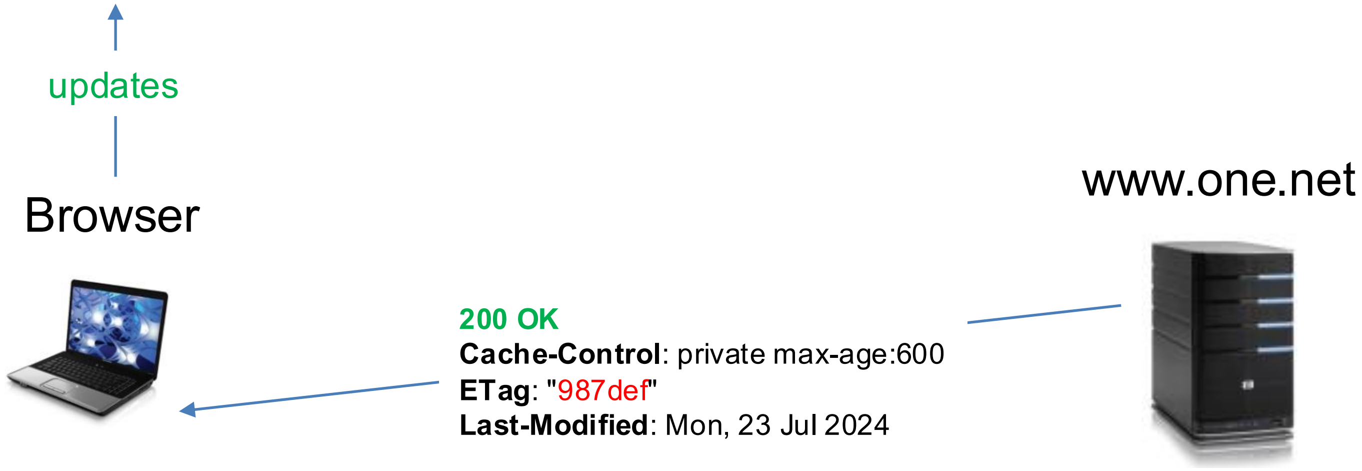
1

If there is no change in the resource, the server responds by 304 status code.
It **does not send** the content of the resource (logo.jpg) to the client.
The browser retrieves the resource from the cache. (not network)
The cache entry is not stale any more, (fresh copy).
It resets max-age counter to 600 seconds.

HTTP Caching

Cache

url	max-age	Etag	Expire	data/cache file
http://www.one.net/logo.jpg	600	987def	23 Jul 2024	/tmp/logo_tmp1234.tmp

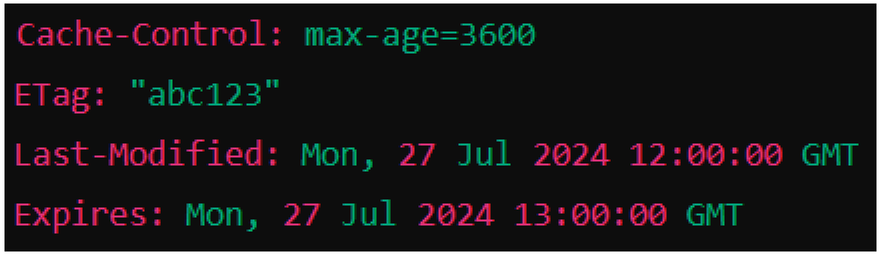


200 OK
Cache-Control: private max-age:600
ETag: "987def"
Last-Modified: Mon, 23 Jul 2024

2

If it changes, it **sends the updated resouce** with new caching headers.
Notice that ETag is different, "987def". (new content means new version)
Last-Modified is the date of last modification of the file.
Notice: Even the file is not modified/edited, its output may be different due to the dynamic content of the backend program.
The cache file is also different.

- "Stale" copy : no longer considered fresh or valid (cached but expired)
  - **Cache-Control: max-age** is expired (600 seconds elapsed)
  - **Expires** header: if the current date and time are past the date and time specified in the "Expires" response header.
  - **Cache-Control: no-cache** : revalidate the cached copy with the server before using it, making the cache potantially stale until it gets confirmation from the server.
  - **ETag and If-None-Match**: the server may use an "ETag" to identify a resource version. If the client's cached version's "ETag" does not match the current "ETag" on the server, the cached copy is stale.
  - **Last-Modified and If-Modified-Since**: If the resource has been modified since the time specified in the "If-Modified-Since" header, the cached copy is stale.
  - **Cache-Control: no-store**: This directive instructs caches not to store any copy of the resouce, making any cached version immediately stale.



```
Cache-Control: max-age=3600
ETag: "abc123"
Last-Modified: Mon, 27 Jul 2024 12:00:00 GMT
Expires: Mon, 27 Jul 2024 13:00:00 GMT
```

- Time-based Expiration:
  - After 3600 seconds, it becomes stale due to max-age
  - If the client checks the "Expires" header, and the current time is past "Mon, 27 Jul 2024 13:00:00 GMT"
- Revalidation:
  - After adding If-None-Match: "abc123" or If-Modified-Since: Mon, 27 Jul 2024 12:00:00 GMT
  - If the server responds with "304 Not Modified", the cached copy is still valid.
  - If the server responds with a new version of the resource, the cached copy is stale and replaced.

Caching Strategies Samples

- Light Caching e.g. html
  - Cache-Control: private, no-cache
  - everytime it validates html pages, it uses the cache version if it is valid. It gets always the latest html.
- Aggressive Caching e.g. CSS, JS, Images
  - Cache-Control: public, max-age: 360000
- No Caching
  - Cache-Control: public, no-store