웹 시스템 설계

Web System Design

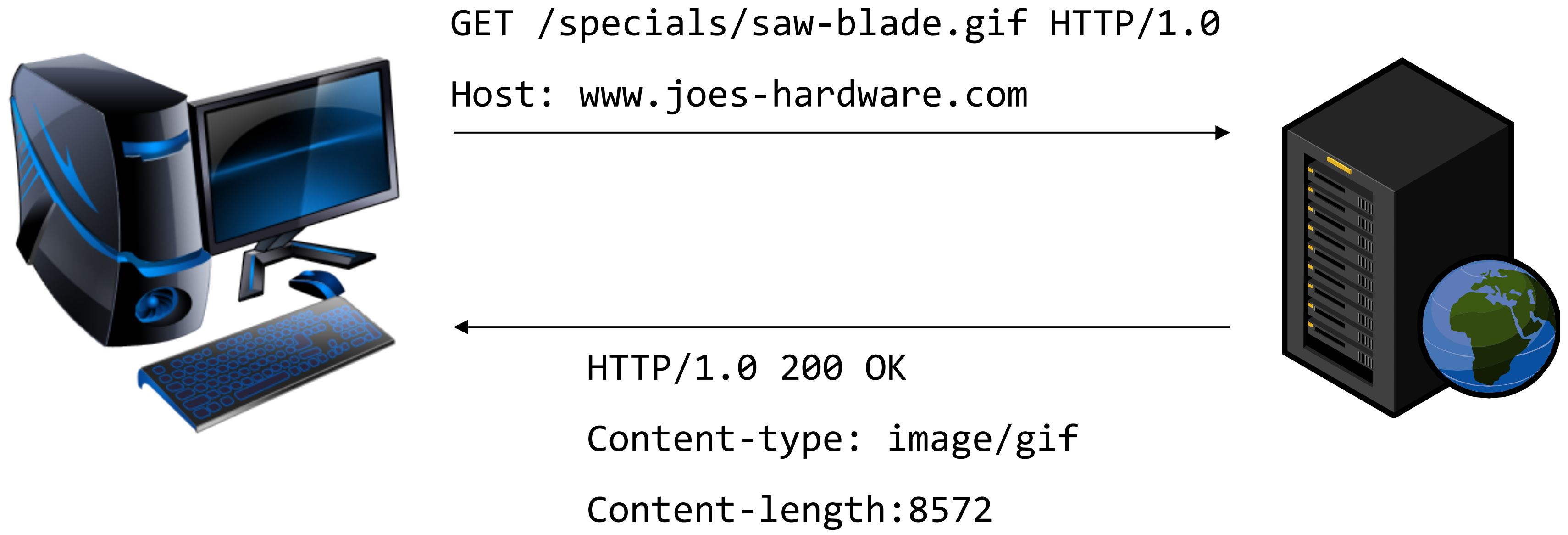
18. HTTP & REST

HTTP Basics

HTTP steps

① browser extracts the server's hostname from the URL.
② browser converts hostname into the IP.
③ browser extracts the port # (if any) from the URL.
④ browser establishes a TCP connection with the server.
⑤ browser sends an HTTP request message to the server.
⑥ server sends an HTTP response back to the browser.
⑦ the connection is closed, and the browser displays the document.

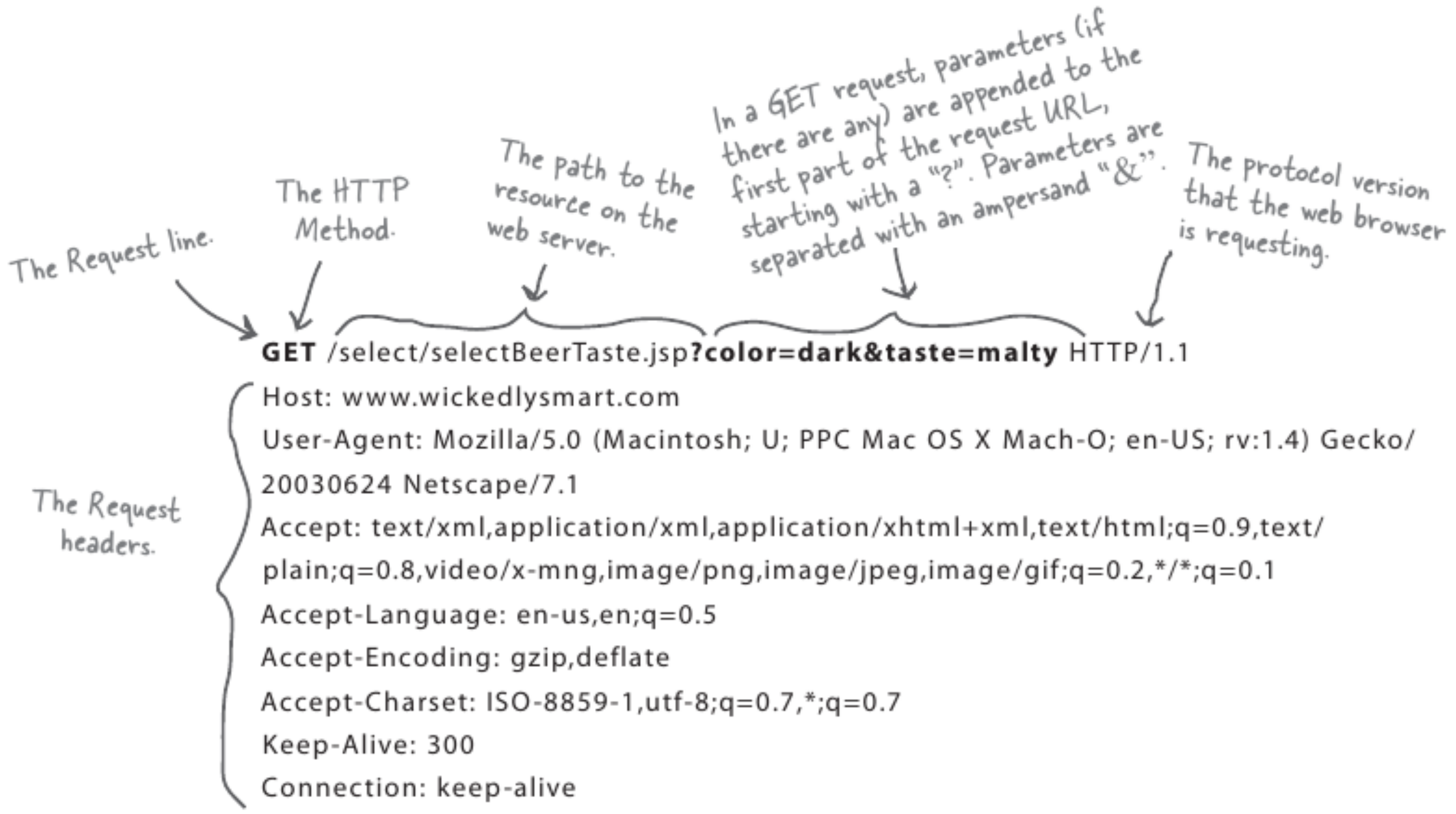
HTTP Transaction



```
GET /specials/saw-blade.gif HTTP/1.0
Host: www.joes-hardware.com
```

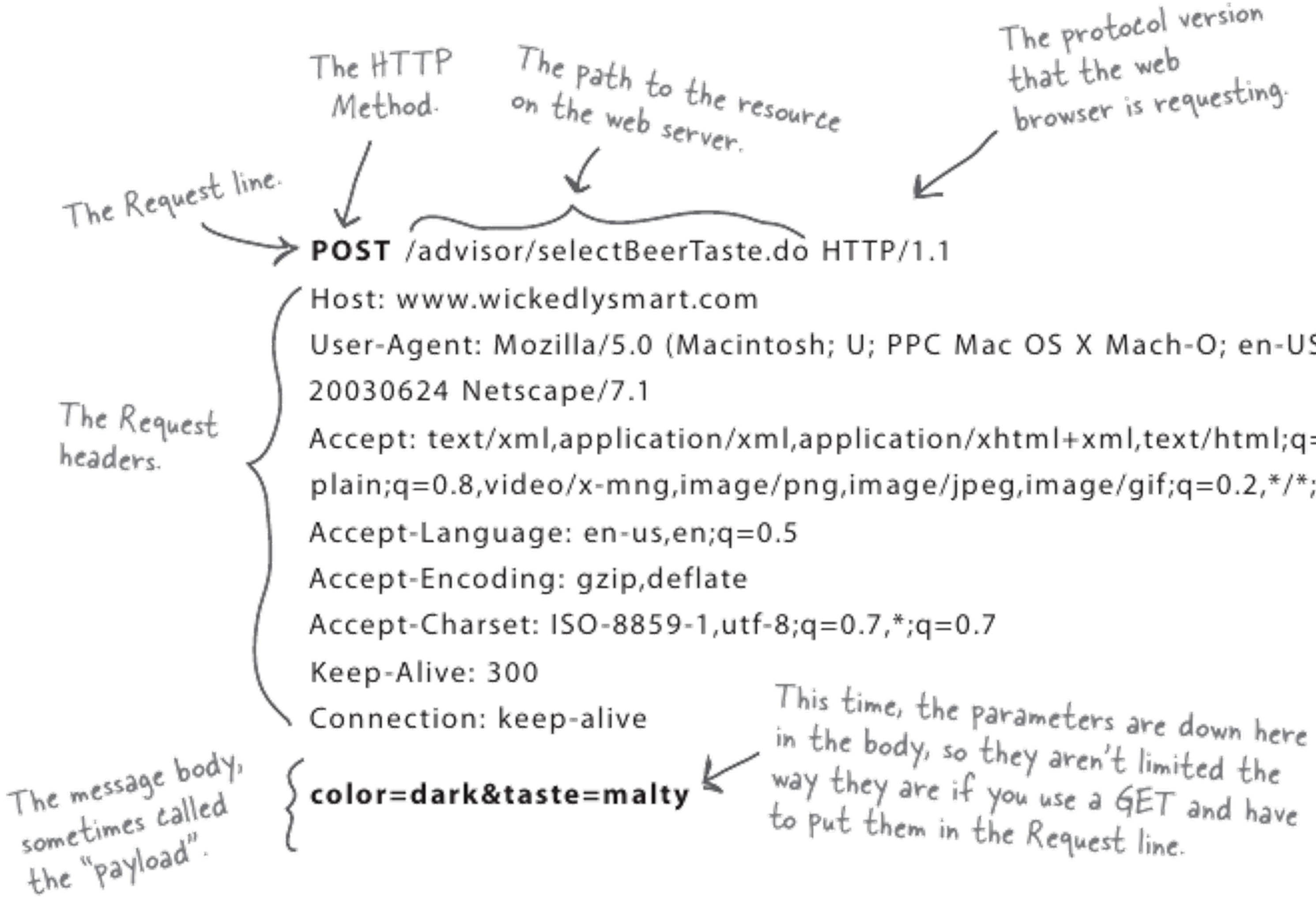
```
HTTP/1.0 200 OK
Content-type: image/gif
Content-length:8572
```

HTTP GET Method


The Request line.
The HTTP Method.
The path to the resource on the web server.
In a GET request, parameters (if there are any) are appended to the first part of the request URL, starting with a "?". Parameters are separated with an ampersand "&".
The protocol version that the web browser is requesting.
The Request headers.

```
GET /select/selectBeerTaste.jsp?color=dark&taste=malty HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/
20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP POST Method

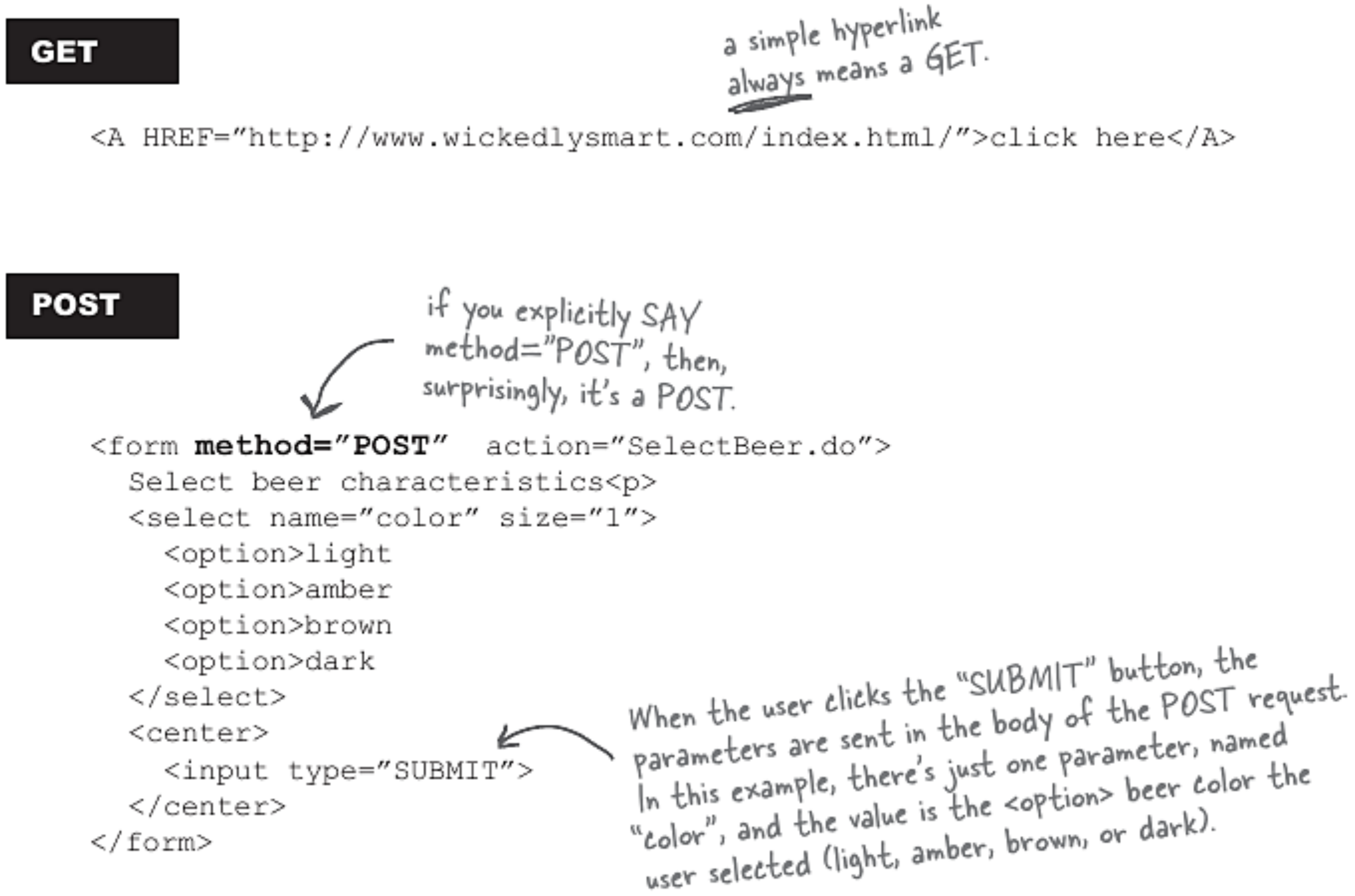

The HTTP Method.
The path to the resource on the web server.
The protocol version that the web browser is requesting.
The Request line.
The Request headers.
The message body, sometimes called the "payload".
This time, the parameters are down here in the body, so they aren't limited the way they are if you use a GET and have to put them in the Request line.

```
POST /advisor/selectBeerTaste.do HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/
20030624 Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

color=dark&taste=malty
```

HTTP GET/POST Method

❖ If there is no method defined, it is **GET**.



GET

a simple hyperlink always means a GET.

```
<A HREF="http://www.wickedlysmart.com/index.html/">click here</A>
```

POST

if you explicitly SAY method="POST", then, surprisingly, it's a POST.

```
<form method="POST"  action="SelectBeer.do">
  Select beer characteristics<p>
  <select name="color" size="1">
    <option>light
    <option>amber
    <option>brown
    <option>dark
  </select>
  <center>
    <input type="SUBMIT">
  </center>
</form>
```

When the user clicks the "SUBMIT" button, the parameters are sent in the body of the POST request. In this example, there's just one parameter, named "color", and the value is the <option> beer color the user selected (light, amber, brown, or dark).

HTTP Transaction

- ❖ HTTP supports several request commands, called HTTP methods.
- ❖ Enough to cover CRUD (create, read, update,. delete)

GET	**send named resource from the server to the client**
POST	**send client data into a server gateway application**
PUT	**store data from client into a named server resource**
DELETE	**delete the named resource from a server**
OPTIONS	send information about the communication options available
CONNECT	for use with a proxy
TRACE	used to invoke a remote, application-layer loop- back of the request message. The final recipient is either the origin server or the first proxy or gateway
HEAD	Send just the HTTP headers from the response

HTTP Request

A Typical HTTP Request

```
GET /search-servlet?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

“Referrer”

Misspelled in the RFC as well as in most implementation

Common - HTTP 1.1 Request Headers (1)

- `Accept`
  - Indicates *`MIME`* types browser can handle
- `Accept-Encoding`
  - Indicates encodings (e.g., *`gzip`* or compress) browser can handle
- `Authorization`
  - User identification for password-protected pages.
  - Preferable: Instead of *`HTTP`* authorization, use *`HTML`* forms to send *`username/password`* and store info in session object. (standard HTTP authorization results in a small, terse dialog box that is unfamiliar to many users.)
  - See more at : https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

Common - HTTP 1.1 Request Headers (2)

- Connection
  - [*HTTP 1.0*] keep-alive → browser can handle persistent connection.
  - [*HTTP 1.1*] persistent connection is default. Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client (e.g., the images associated with a page, or cells within a framed page).
  - Server Scripts can't do this unilaterally; the best they can do is to give the server enough info to permit persistent connections.
- Cookie
  - Gives cookies previously sent to client.

Common - HTTP 1.1 Request Headers (3)

❖ `Host`

- Indicates host given in original URL
- This is a required header in *HTTP 1.1*. This fact is important to know if you write a custom *HTTP* client or telnet to a server and use the *HTTP/1.1* version.
- 자원을 받기를 원하는 host name

❖ `If-Modified-Since`

- Indicates client wants page only if it has been changed after specified date
- 304 (Not modified) → no newer result is available

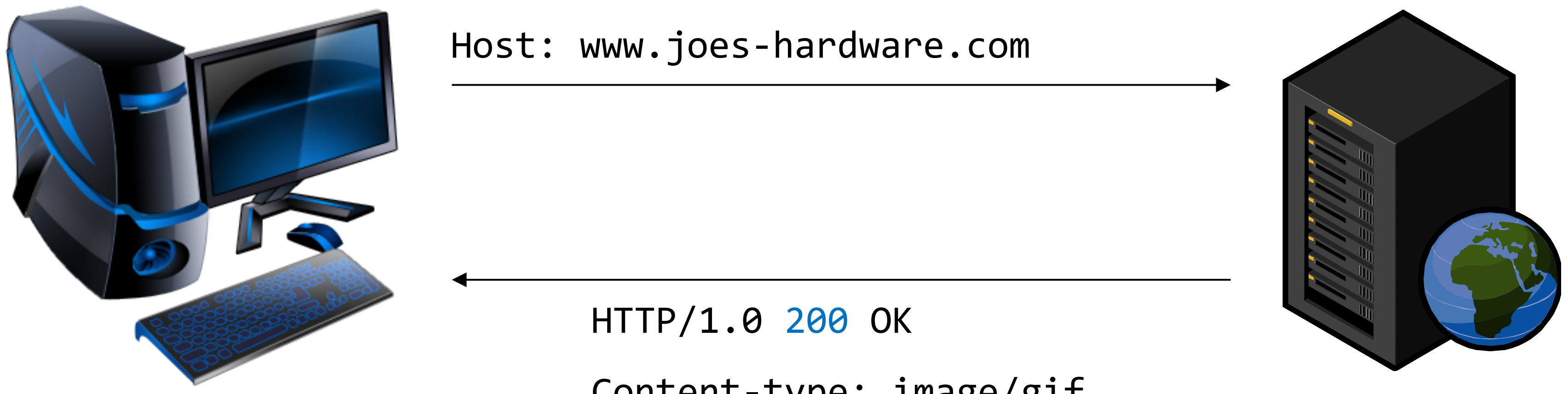
❖ `Referer`

- webPage1 → click → webPage2
- request webPage2 includes webPage1 as referer (유입 경로 추적)

❖ `User-Agent`

Status Codes

HTTP Request/Response



```
Get /specials/saw-blade.gif HTTP/1.0
Host: www.joes-hardware.com
```

```
HTTP/1.0 200 OK
Content-type: image/gif
Content-length:8572
```

Informational 1xx
Success 2xx
Redirection 3xx
Client Error 4xx
Server Error 5xx

Common HTTP 1.1 Status Codes

- 200 (OK)
  - Everything is fine; document follows.
  - `(Usually) Default` option.

- 204 (No Content)
  - Browser should keep displaying previous document.
  - no new doc. Useful when client periodically reloads a page

- 301 (Moved Permanently)
  - Requested document permanently moved elsewhere (indicated in Location header).
  - Browsers go to new location automatically.
  - Browsers are technically supposed to follow 301 and 302 (next page) requests only when the incoming request is *GET*, but do it for *POST* with 303. Either way, the Location URL is retrieved with *GET*.
  - Location header에 새 경로 전달 → browser should automatically follow the link to the new URL

Common HTTP 1.1 Status Codes

- 302 (Found)
  - Requested document temporarily moved elsewhere (indicated in Location header).
  - Server Script should Redirect, not only sending the status.
  - Browsers go to new location automatically.

- 401 (Unauthorized)
  - Browser tried to access password-protected page without proper Authorization header.

- 404 (Not Found)
  - No such page. Server Script should set actions for this.
  - Problem:
    - Internet Explorer and small (< 512 bytes) error pages.
    - IE ignores small error page by default.

HTTP Response

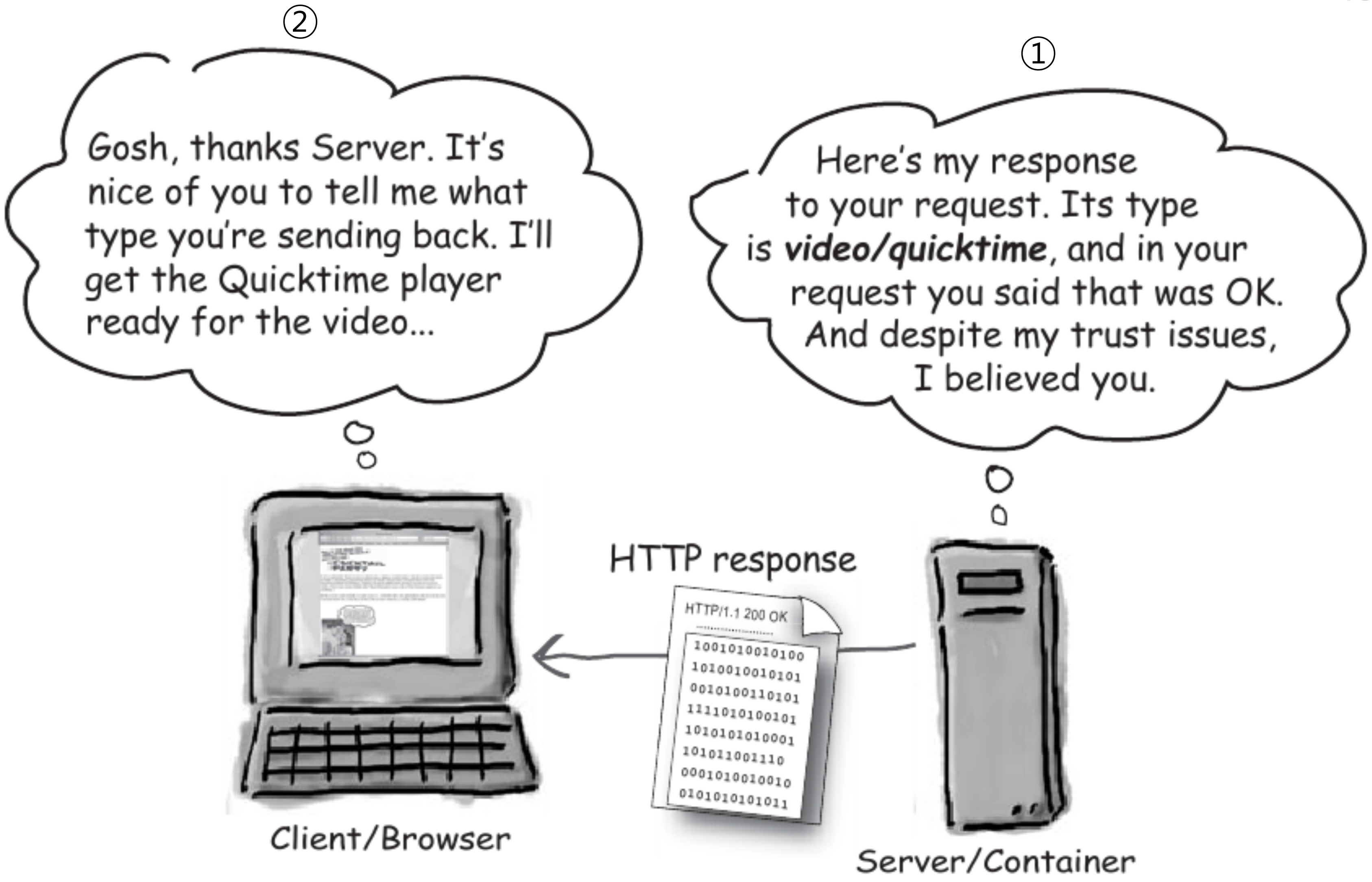
HTTP Request/Response

Typical request

```
GET /servlet/SomeName HTTP/1.1
Host: ...
Header2: ...
...
HeaderN:
  (Blank Line)
```

Typical response

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
    (Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY>
</HTML>
```


②
Gosh, thanks Server. It's nice of you to tell me what type you're sending back. I'll get the Quicktime player ready for the video...
①
Here's my response to your request. Its type is ***video/quicktime***, and in your request you said that was OK. And despite my trust issues, I believed you.
HTTP response
HTTP/1.1 200 OK
1001010010100
1010010010101
0010100110101
1111010100101
1010101010001
101011001110
0001010010010
0101010101011
Client/Browser
Server/Container

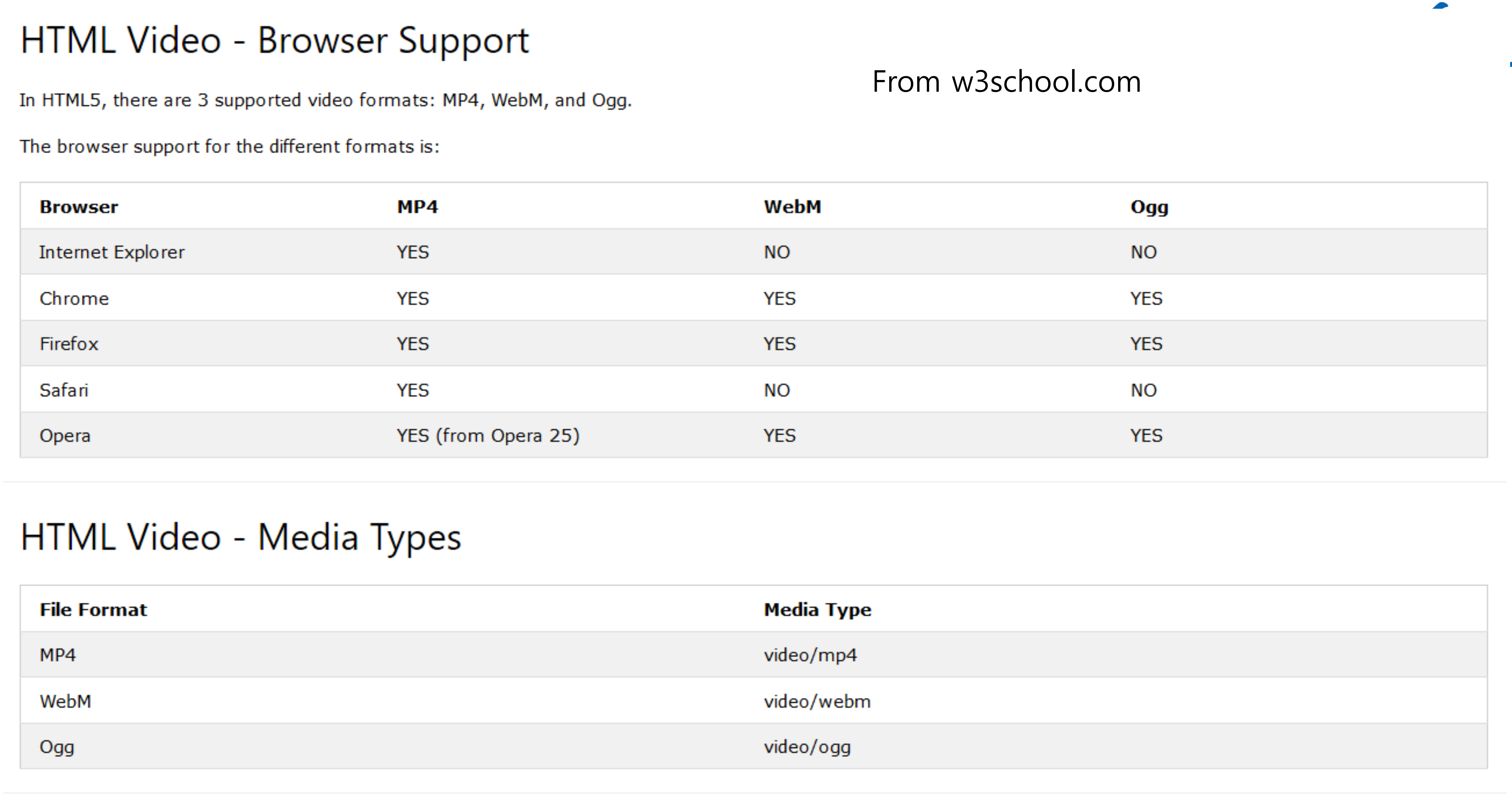
Common MIME Type (Media type): identification for file formats

`application/msword`	Microsoft Word document
`application/octet-stream`	Unrecognized or binary data
`application/pdf`	Acrobat (.pdf) file
`application/postscript`	PostScript file
`application/vnd.ms-excel`	Excel spreadsheet
`application/vnd.ms-powerpoint`	Powerpoint presentation
`application/x-gzip`	Gzip archive
`application/x-java-archive`	JAR file
`application/x-java-vm`	Java bytecode (.class) file
`application/zip`	Zip archive
`audio/basic`	Sound file in .au or .snd format
`audio/x-aiff`	AIFF sound file

`audio/x-wav`	Microsoft Windows sound file
`audio/midi`	MIDI sound file
`text/css`	HTML cascading style sheet
`text/html`	HTML document
`text/plain`	Plain text
`text/xml`	XML document
`image/gif`	GIF image
`image/jpeg`	JPEG image
`image/png`	PNG image
`image/tiff`	TIFF image
`video/mpeg`	MPEG video clip
`video/quicktime`	QuickTime video clip

MIME (Multipurpose Internet Mail Extensions)
Content-type of MIME

From w3school.com



HTML Video - Browser Support

In HTML5, there are 3 supported video formats: MP4, WebM, and Ogg.

The browser support for the different formats is:

Browser	MP4	WebM	Ogg
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	NO	NO
Opera	YES (from Opera 25)	YES	YES

HTML Video - Media Types

File Format	Media Type
MP4	video/mp4
WebM	video/webm
Ogg	video/ogg

Common HTTP 1.1 Response Headers (1)

❖ `Cache-Control` (HTTP 1.1) and `Pragma` (HTTP 1.0)
- A *`no-cache`* value prevents browsers from caching page.
- *`public, private, no-store`*

❖ `Content-Disposition`
- Lets you request that the browser ask the user to save the response to disk in a file of the given name ("File Download")

  `Content-Disposition: attachment; filename=file-name`

❖ `Content-Encoding`
- The way document is encoded. (전송 중 인코딩 형식) e.g., gzip

❖ `Content-Length`
- The number of bytes in the response.

Common HTTP 1.1 Response Headers (2)

- `Content-Type`
  - The *MIME* type of the document being returned.
- `Expires`
  - The time at which document should be considered out-of-date and thus should no longer be cached.
- `Last-Modified`
  - The time document was last changed.
  - Don't set this header explicitly; use a framework provided function instead.

Common HTTP 1.1 Response Headers (3)

❖ `Location`
- The URL to which browser should reconnect.
- To ask a browser to load a different page (3xx) and to provide information about the location of newly created resource (2xx)

❖ `Refresh`
- The number of seconds until browser should reload page. Can also include *URL* to connect to.

  <META HTTP-EQUIV="Refresh" CONTENT="5; URL=http://host/path/">

❖ `Set-Cookie`
- The cookies that browser should remember. Don’t set this header directly.

❖ `WWW-Authenticate`
- The authorization type and realm needed in Authorization header.

Web Server and API (or application) Server

Descriptions

- ❖ You can talk to servers over HTTP or HTTPS

- ❖ Web Servers:
  - Serve (static) HTML pages to fulfill request from clients (POST and GET method)

- ❖ API Servers:
  - Instead of generating HTML pages or any markups, it generates data (mostly JSON)
  - Fundamental job is to provide its client with access to business logic that generates dynamic content.
  - Use more variety of HTTP verbs like POST, GET, DELETE, PUT (RESTFul)

- ❖ Express, Hapi, and Koa are all working as both Web Servers and Application Servers

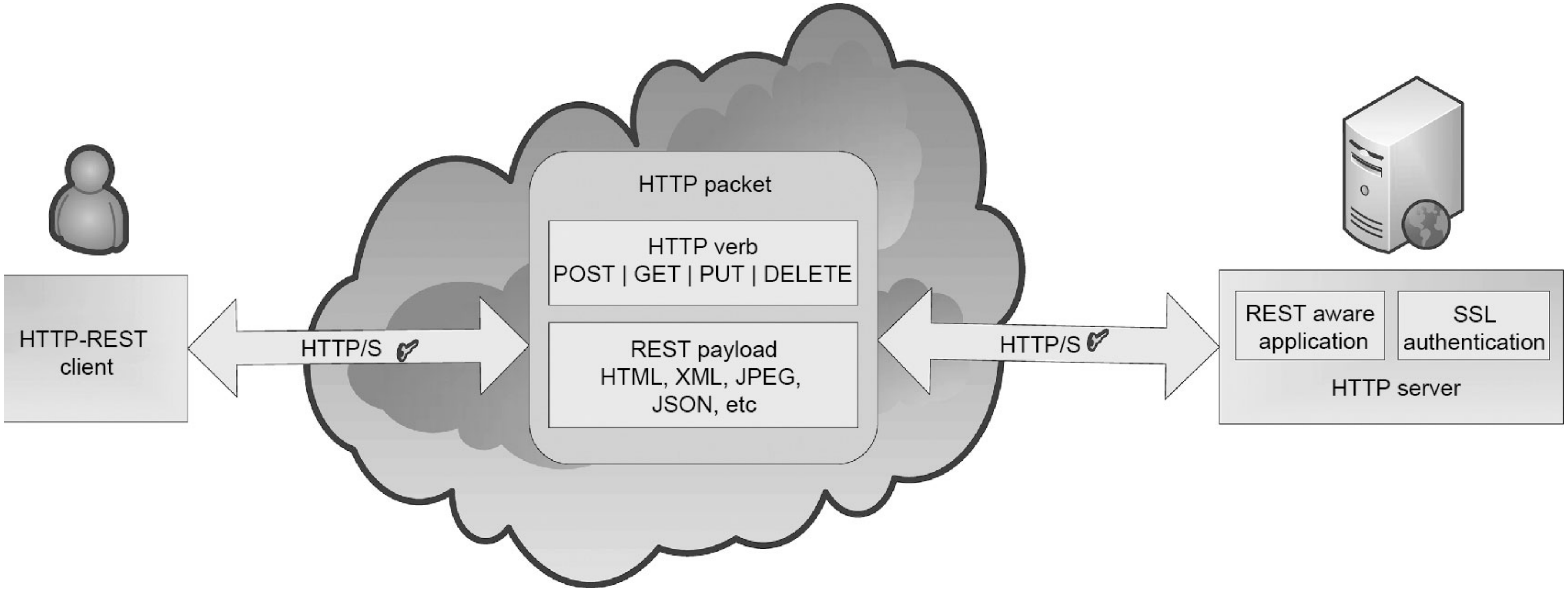
Simple Web Server with RESTful style

REpresentational State Transfer

- Architectural style that is defined by Dr. Roy Fielding (one of principle authors of the HTTP specification and co-founder of Apache HTTP server project) in his doctoral dissertation in 2000
  - A software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web.
  - Used by many mega enterprises such as Google, Amazon, Yahoo!, Facebook and Twitter

- Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures" (PhD diss, University of California, Irvine, 2000), www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Fielding analyzed the success of WWW and defined the REST based on the architectural styles of successful web services of WWW

REST – A software tool for distributed systems



Four principles of RESTful

- ❖ Resource Identification through URIs:
    - RESTful web services exposes a set of resources which identify targets of interaction. Resources are identified by URI (Uniform Resource Identifier)
- ❖ Uniform, Constrained Interface
    - Interaction with RESTful web services is done via the HTTP protocol.
    - Resources are manipulated using a fixed set of four CRUD (create, read, update and delete) → PUT, GET, POST, DELETE
- ❖ Self-descriptive Message
    - A REST message includes enough information to describe how to process message.
    - In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.)
- ❖ Stateless interactions
    - Interactions are stateless

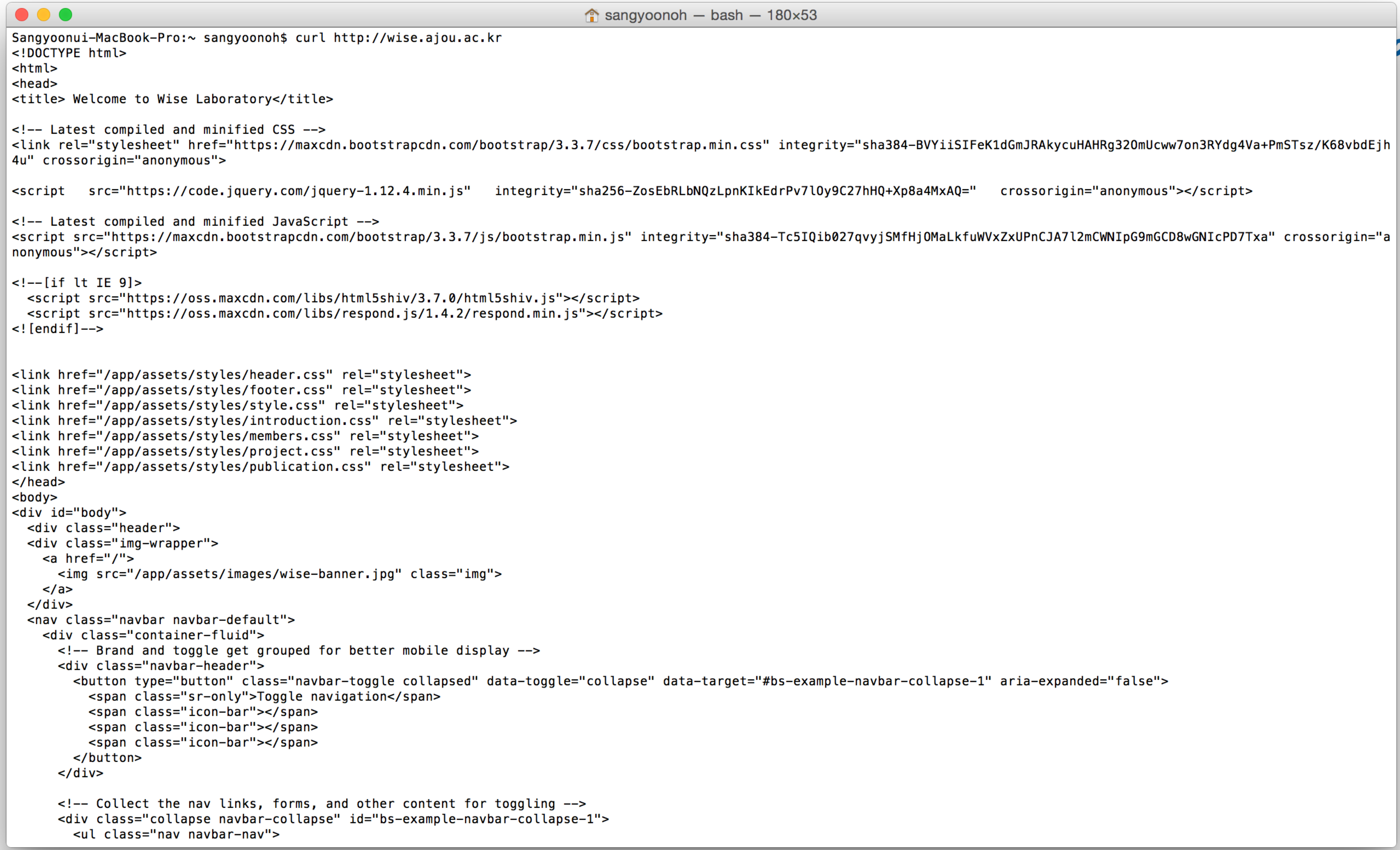
Example of REST interface (http://spoqa.github.io/2012/02/27/rest-introduction.html)

HTTP Verb	Path	action	used for
GET	/photos	index	display a list of all photos
GET	/photos/new	new	return an HTML form for creating a new photo
POST	/photos	create	create a new photo
GET	/photos/:id	show	display a specific photo
GET	/photos/:id/edit	edit	return an HTML form for editing a photo
PUT	/photos/:id	update	update a specific photo
DELETE	/photos/:id	destroy	delete a specific photo

To-Do list Example with cURL

- POST — Add items to the to-do list
- GET — Display a listing of the current items, or display the details of a specific item
- DELETE — Remove items from the to-do list
- PUT — Should modify existing items (we'll skip PUT)

- cURL (http://curl.haxx.se/download.html) **command line tool and library** for transferring data with URLs
- a powerful command-line HTTP client that can be used to send requests to a target server.
  - in place of a web browser, to interact with your web service
  - curl [options] targetURL
    - -d, –data <data>



sangyoonoh — bash — 180×53

```
Sangyoonui-MacBook-Pro:~ sangyoonoh$ curl http://wise.ajou.ac.kr
<!DOCTYPE html>
<html>
<head>
<title> Welcome to Wise Laboratory</title>

<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh
4u" crossorigin="anonymous">

<script   src="https://code.jquery.com/jquery-1.12.4.min.js"   integrity="sha256-ZosEbRLbNQzLpnKIkEdrPv7lOy9C27hHQ+Xp8a4MxAQ="   crossorigin="anonymous"></script>

<!-- Latest compiled and minified JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="a
nonymous"></script>

<!--[if lt IE 9]>
  <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
  <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js"></script>
<![endif]-->


<link href="/app/assets/styles/header.css" rel="stylesheet">
<link href="/app/assets/styles/footer.css" rel="stylesheet">
<link href="/app/assets/styles/style.css" rel="stylesheet">
<link href="/app/assets/styles/introduction.css" rel="stylesheet">
<link href="/app/assets/styles/members.css" rel="stylesheet">
<link href="/app/assets/styles/project.css" rel="stylesheet">
<link href="/app/assets/styles/publication.css" rel="stylesheet">
</head>
<body>
<div id="body">
  <div class="header">
  <div class="img-wrapper">
    <a href="/">
      <img src="/app/assets/images/wise-banner.jpg" class="img">
    </a>
  </div>
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -->
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
      </div>

      <!-- Collect the nav links, forms, and other content for toggling -->
      <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
        <ul class="nav navbar-nav">
```

Creating Resource

- `POST` to create an entry in the to-do list
  - Get the used HTTP method by checking `req.method`

- Node's HTTP parser reads in and parses request data
  - **that data is available in the form of `data` events that contain chunks of parsed data ready to be handled by the program**

Creating Resource

```
const http = require('http');
const url = require('url');
const items = [];                                      // array in memory

const server = http.createServer( function(req, res) {
    if (req.method === 'POST') {
       var item = '';                                  // buffer for the incoming
       req.setEncoding('utf8');
       req.on('data', function (chunk) { // EventEmitter
            item += chunk;
       });
       req.on('end', function () {
            items.push(item);                          // push to array
       });
       res.end('Okay\n');
}
```

Fetching resources with GET requests

```
    else if (req.method === 'GET') {
        var item = '';                              // buffer for the incoming
        items.forEach(function (item, i) {
            res.write(i + ') ' + item + '\n');
        });
        res.end();
    }
}).listen(1337, "127.0.0.1");
```

```
$ curl -d 'buy groceries' http://localhost:1337
OK
$ curl -d 'buy node in action' http://localhost:1337
OK
$ curl http://localhost:1337
0) buy groceries
1) buy node in action
```

Removing resources with DELETE requests

- To delete a item
  - need to check the requested URL, which is how the HTTP client will specify which item to remove.
  - In this case, the identifier will be the array index in the items array;
    for example, `DELETE /1` or `DELETE /5`.
- Convert `ID` to a number
  - with `String#slice()` method, which returns a portion of the string between two indexes
  - convert this string to a number, it can be passed to the JavaScript global function `parseInt()`, which returns a Number.
- `parseInt()`
  - `http://www.w3schools.com/jsref/jsref_parseint.asp`
- Array splice
  - method adds/removes items to/from an array, and returns the removed item(s)
  - `https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/splice`

Removing resources requests

```
else {
          const path = url.parse(req.url).pathname;
          const i = parseInt(path.slice(1), 10);
          var item = '';                                    // buffer for the incoming
          if (isNaN(i)) {
               res.statusCode = 400;          // I don't understand
               res.end('Invalid item id');
          } else if (!items[i]) {
               res.statusCode = 404;          // Not found
               res.end('Item not found');
          } else {
               items.splice(i, 1);
               res.end('Okay\n');
          }
     }
}).listen(1337);
```

Problems of the current approach

❖ Hard to handle route (url + method) ← routing

❖ Hard to build applications in organized way
  - easy fetching of static files
  - Template-ing

❖ Hard to apply architecture