CSE1500 Web Technology Notes

Berken Tekin

January 4, 2022

Contents

1 DISCLAIMER

Most of the notes here are a summarization of the provided lectures and lecture notes. When in doubt, always refer to the original sources.

2 HTTP

2.1 Status Codes

1xx informational response Indicates that the request was received and understood.

2xx success The action requested by the client was received, understood and accepted.

3xx redirection Indicates the client must take additional action to complete the request. Many of these status codes are used in URL redirection.

4xx client errors For errors that seem to have been caused by the client.

5xx server errors The server failed to fulfil a request. Except when responding to a HEAD request, the server *should* include an entity containing an explanation of the error situation, and indicate whether it is a temporary or permanent condition. Likewise, user agents should display any included entity to the user.

2.2 Common Headers

Header	Description
Content-Type	Entity Type
Content-Length	Length/size of the message
Content-Encoding	Data transformations applied to the entity
Content-Location	Alternative location of the entity
Content-Range	Range defines the pieces sent for partial entities
Content-MD5	Checksum of the content
Expires	Date at wich the entity wll become stale
Last-Modified	Most recent creation/modification date of the entry
Allow	The legal request methods for the entity
Connection & Upgrade	Protocol Update

2.2.1 Content-Type

- MIME$_{\text{Multipurpose Internet Mail Extensions}}$ types determine the clients' reaction to data.
- A standard MIME Pattern looks like `[primary object type]/[subtype]` Every MIME has a primary object type and a subtype.
- Examples:
  - text/plain
  - text/html
  - image/jpeg
  - application/pdf

2.2.2 Content-Length

- Indicates the *size* of the entity body
- Necessary to detect premature message truncation due to extenuating circumstances
- Used to discover where one HTTP message ends and the next begins for **persistent connections**, which reuse a TCP connection for multiple HTTP request/response messages

2.2.3 Content-MD5 (RFC 1321)

- HTTP messages are sent via TCP (this'll change in HTTP/3)
- However, as the internet is decentralize, different servers implement the protocol differently, which causes bugs
- To counter this, sender generates a 128-bit MD5 checksum of the content to detect unintended modifications. This procedure is called **sanity check.**
- Has been removed from HTTP/1.1 specification (2014), however; this simple technique is still in use

2.2.4 Expires

- Contains the date/time after which the response is considered stale/invalid.
- Invalid dates, like the value 0, represent a date in the past and mean that the resource is already expired.
- Web caches have several advantages:
  1. Reduces redundant data transfer
  2. Reduces network bottlenecks
  3. Reduces demand on origin servers
  4. Reduces distance delay
- `Cache-Control` is a header that overrules `Expires`. While `Expires` indicates a resource's expiration date in absolute terms (a specific date/time), `Cache-Control` indicates it in relative terms (seconds sicne being sent). This is an advantage because the server doesn't have to "reaassign" a specific date/time for expiration repeatedly.

2.2.5 Last-Modified

- Contains the date on which the resource was last **altered**
- No indication about the amount of changes
- Often used with `If-Modified-Since` for cache revalidation requests so that the origin server only returns the documents if it has been modified since the given date

2.2.6 Connection & Upgrade

- When using `HTTP/1.1` the client always initiates the connection
- Within this boundary there are ways to simulate a **server-side push** of data:

  Polling: client regularly sends HTTP requests to receive updates
  - Wastes bandwidth

  Long Polling: client sends an HTTP request and the server holds it open until new data arrives
  - Difficult to implement
  - Not widely used

- This header is used to switch protocols to circumvent these restrictions and make server-side push more efficient with protocols such as WebSocket through the following procedures:

  1. Client and server have to agree to the protocol upgrade.
  2. Client initiates the upgrade with two request headers:
     `Connection:Upgrade`
     `Upgrade:[protocols]`
  3. Server responds with a `101 Switching Protocols` status if such upgrade is possible.
  4. Once established, both the client and the server can push data.

2.3 Common Methods

Method	Definition
GET	Get a document from the Web server
HEAD	Get the header of a document from the Web server
POST	Send data from the client to the server for processing
PUT	Save the body of the request on the server
TRACE	Trace the message through the proxy servers to the server
OPTIONS	Determine what methods can operate on a server
DELETE	Remove a document from a Web server

2.4 Telnet

2.4.1 Properties

1. Interactive text-oriented communication with a server

2. Uses TCP to communicate with server
3. Won't work with HTTPS
4. Connections are not encrypted

2.4.2 Commands

1. Connecting to a server and requesting headers:

```
$ telnet www.reddit.com 80
HEAD /r/TUDelft HTTP/1.1
Host: www.reddit.com
Connection: close
```

This gives a `301 Moved Permanently` error because reddit uses `HTTPS` for security reasons.

Full message:

```
HTTP/1.1 301 Moved Permanently
Retry-After: 0
Location: https://www.reddit.com/r/TUDelft
Content-Length: 0
Accept-Ranges: bytes
Date: Tue, 14 Dec 2021 00:04:13 GMT
Via: 1.1 varnish
Connection: close
Cache-Control: private, max-age=3600
Strict-Transport-Security: max-age=31536000; includeSubdomains
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: snooserv
X-Clacks-Overhead: GNU Terry Pratchett
```

Therefore, we need a different tool for HTTPS.

2.5 OpenSSL

2.5.1 Properties

1. Its `s_client` component works just like telnet, though it's a bit less interactive

2. Uses SSL+TLS for secure connection
3. Supports HTTPS
4. Connections are encrypted and secure

2.5.2 Commands

1. Command-line args:

   -crlf Translates a $LF_{\text{Line Feed}}$ (11th ASCII character) into CR+LF

   -connect Connects to a SSL HTTP server.

2. Connecting to a server and requesting headers:

```
$ openssl s_client -crlf -connect www.reddit.com:443
HEAD /r/TUDelft HTTP/1.1
Host: www.reddit.com
Connection: close
```

   Output:

```
HTTP/1.1 200 OK
Connection: close
Cache-control: private, s-maxage=0, max-age=0, must-revalidate, no-store
Content-Type: text/html; charset=utf-8
Accept-Ranges: bytes
Date: Tue, 14 Dec 2021 00:21:09 GMT
Via: 1.1 varnish
Vary: Accept-Encoding
Set-Cookie: loid=0000000000hjgnftu8.2.1639441268721.Z0FBQUFBQmh0LU4wem9QRHBnYUhVQ1
Set-Cookie: session_tracker=ibekfgdqdfkmfrhhcd.0.1639441269865.Z0FBQUFBQmh0LU4xeDh
Set-Cookie: token_v2=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MzkONDQ3NDgs
Set-Cookie: csv=2; Max-Age=63072000; Domain=.reddit.com; Path=/; Secure; SameSite=
Set-Cookie: edgebucket=MdiF5bpSjBeHhPKevt; Domain=reddit.com; Max-Age=63071999; Pa
Strict-Transport-Security: max-age=31536000; includeSubdomains
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: snooserv
X-Clacks-Overhead: GNU Terry Pratchett
```

2.6 URL

- Uniform Resource Locators offer a standardized way to point to a resource on the Internet
- Not restricted to HTTP, URLs support different schemes/protocols such as `HTTP`, `HTTPS`, `mailto`, `file`, `ftp` etc.

2.6.1 Syntax

`<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`

`<scheme>` determines the protocol to use when connecting to the server

`<user>:<password>` is the username and/or password to access a protected resource

`<host>` is the domain name or IP address of the server

`<port>` is the port on which the server is expecting requests

`<path>` is the local path to the resource

`<params>` are additional input parameters applications may require

`<query>` are parameters passed to gateway resources (e.g. a search engine) Common convention: `name1=value1&name2=value2...`

`<frag>` the name of a piece of a resource (i.e. a part of a page), only used by the client. For example, with this tag the middle of a webpage may be shown by default. However, the client will always retrieve the entire page

2.6.2 Design Restrictions

1. No invisible/non-printing characters
2. Initially restricted to ASCII$_{\text{American Standard Code for Information Interchange}}$ characters, biased towards English speakers
   - Added Later: character encoding e.g. whitespace -> %20
   - Punycode (RFC 3492) is used to **uniquely** and **reversibly** transform a Unicode string into an ASCII string. Introduces a potential security issue in *mixed* scripts.

2.6.3 Weaknesses

1. URLs point to a location instead of a Web resource. When the location of a website changes, the old URL won't work anymore.

2.7 Authentication

HTTP is an **anonymous**, **stateless** request/response protocol. The same request, sent by different clients, is treated in exactly the same manner. Now, there are different identification methods such as:

1. HTTP headers
2. Client IP address tracking
3. Fat URLs
    - Track users through the generation of unique URLs
        (a) First time a user visits a resource within a Website, a **unique ID** is generated by the server
        (b) Server redirects client to the fat URL (URL + unique ID)
        (c) Server **rewrites the HTML** when a HTTP request with a fat URL is received (by adding ID to all hyperlinks)
        (d) As a result, independent HTTP requests are tied into a single session.
    - Issues:
        – Fat URLs are ugly
        – They cannot be shared
        – They break web caching mechanisms
        – Extra server load through HTML page rewrites
        – The ID is lost when the user navigates away from the website
4. User login (HTTP Basic Authentication)
    - Server explicitly asks the user for authentication (401 Login Required)
    - HTTP has a **built-in mechanism** to support username/password based authentication via `WWW-Authenticate` and `Authorization` headers
        – Username and passwords are joined together by a colon and converted to **base-64 encoding**

  - Base-64 ensures that only HTTP compatible characters are entered into a message.
- HTTP is **stateless:** Once logged in, the client sends the login information with each request.
- Issues:
  (a) Username and password can be decoded trivially, the data is not encrypted. HTTPS solves this issue by encrypting sent data.
  (b) Users tend to reuse login/password combinations

2.8 Security

- Secure HTTP should provide:

  Server Authentication Client is sure to talk to the right server

  Client authentication Server is sure to talk to the right client

  Integrity Client and server are sure that their data is intact

  Encryption The data is sufficiently encrypted

  Efficiency Providing security should be a reasonable endeavour

- `HTTPS` is the most popular, secure form of HTTP
  - URL Scheme is `https://` instead of `http://`
  - Request and response data are encrypted before being sent across the network via $\text{SSL}_{\text{Secure Sockets Layers}}$. Client and server *negotiate* the cryptographic protocol to use.
  - TRIVIA: To use `HTTPS` on your website, you need a `TLS` certificate from a $\text{CA}_{\text{Certificate Authority}}$. Let's Encrypt provides this service free of charge.

3 HTML5

3.1 Overview

- HTML 5 is a set of related technologies that together enable rich web content
- Successor to XHTML and HTML 4.01

3.1.1 Features

Core HTML5 marks up content

CSS controls the appearance of marked-up content

JavaScript manipulates the contents of HTML documents & responds to user interactions

Not all browsers support all features.

4 JavaScript

4.1 What is JavaScript?

- JavaScript is an interpreted programming language designed to implement complex, interactive features on web pages. However, JS is also used in other areas such as micro-controllers.
- JavaScript adheres to the ECMAScript standard.
- JavaScript is a dynamic language, meaning you can't enforce a certain *type* on a variable. All variables can hold any type.

4.2 Scripting

- Scripts can be applied to the context of a website in two ways: Server-side scripting and client-side scripting.

  Server-side scripting refers to scripts that run on the web server. Only the results of the scripts are returned to the client.
  - Advantage: The results are returned in plain HTML, so the computational power of the client platform is irrelevant.
  - Disadvantage: As all computations are conducted on the server, this may result in an increasing server load.

  Client-side scripting sends the script itself (and relevant data if necessary) to the client, who executes the code themselves.
  - Advantage: The only job of the server is to send the script and data to the client to be processed, which reduces server load.

  - Disadvantage: The performance of the web application is dependent on the client, as they do the hard work by executing the script.

- The lecturer recommends the `<script>` tag to be put to the bottom of the `<body>`, however; apparently there are more modern ways.

4.3 Functional Programming

- JS allows functions to be treated as data, in other words it supports functional programming by treating functions as first-class citizens.
  - TRIVIA: The first chapter of SICP is an amazing introduction to functional programming, you may skim over it if you have the time.

4.3.1 Function as data: an example

- Let's observe this following code (which you can find at the official lecture notes):

```
function toPrint(x) {
  console.log(x);
}

function my_func(x, y) {
  y(x);
}

my_func(5, toPrint);
```

RESULTS:

```
5
undefined
```

This is a perfect example of functional programming.

  - As you can see, `my_func` takes two arguments `x` and `y` and *applies* `y` to `x`. It is implicitly assumed that `y` is a function: using another data type would result in a `TypeError`.

- The first result is produced by applying `y`, in our case `toPrint`, to `x`. `console.log(x)` prints the value of `x`, `5`.
- The second result is the value `my_func` *returns*, in our case it doesn't have a return value. So `undefined` is printed.

4.4 Scoping, hoisting and `this`

4.4.1 Scoping

- It is not always possible to access to a variable everywhere inside the code. For example, in Java you can't access a `private` value outside of you class, that's why we have `getters` and `setters`. They're only visible in the **scope** of your class. Thankfully, scoping is not as complicated in JavaScript.

- JavaScript has very few scopes: `local`, `global` and `block` (introduced with ES6).

  Local/function scope is the scope of the function. Variables declared inside a function cannot be accessed elsewhere. Local variables only exist in the context of the function. Each function creates a new scope.[source]

  Global scope includes the whole program. A variable declared outside of the function becomes global. It's globally accessible.

  Block scope is provided by the `let` and `const` keywords (both of which also introduced with ES6). Blocks are delineated with curly braces `{ }`. So, a variable declared this way:

```
1 {
2     let x = 2;
3 }
```

  CANNOT be accessed outside of the curly braces. However, variables declared with `var` cannot have block scope. So:

```
1 {
2     var x = 2;
3 }
```

  CAN be accessed outside of the curly braces if they do not belong to a function.

- Here's a table of every scope in ES6 JavaScript[source]:

Where/how	Scope
`var` declared within a function	local
`var` declared outside of a function	global
`let` (ES6)	block
`const` (ES6)	block
variable declaration without `var/let/const`	global

4.4.2 Hoisting

Hoisting allows functions to be safely used in code *before* they are declared.

Variables can also be hoisted, however; JS does not hoist initializations Take a look at the following code:

```
// Returns undefined from hoisted var declaration (not 6)
console.log(num);

// Declaration and initialization
var num = 6;

// Returns 6 after the line with initialization is executed.
console.log(num);
```

- This rule applies for function expressions as well. If you define a variable as a function, the expression won't be hoisted.
- Variable/function *declarations* are hoisted:

```
f();
console.log(x); // 5
console.log(y); // 3
function f() { // function declaration
            x = 5; // global scope
            y = 3; // global scope
}
```

4.4.3 `this`

- In Java, `this` refers to the current object. However, in JS what `this` refers to is dependent on how the function containing `this` was called.

- The bind keyword can be used to independently set the function's `this` value.

4.5 Design Patterns

- Instead of trying to come up with novel ways to do a job, we can use tried and tested, effective design patterns for certain tasks.

4.5.1 Objects

- In JavaScript, functions are objects.

4.5.2 Creation, modification and access to objects

- There are several ways to create, modify and access objects:

```
var game = new Object();
game["id"] = 1;
game["player1"] = "Alice"; //bracket notation
game.player2 = "Bob"; //dot notation
console.log(game["player2"]); //prints out "Bob"
console.log(game.player1); //prints out "Alice"

game["won lost"] = "1 12"; // Can't be accessed using dot notation

game.printID = function () {
console.log(this.id);
};
game["printID"](); // prints out "1"
game.printID(); //prints out "1"
```

- Objects can also be created using **object literals:**

```
var game = {
  id: 1,
  player1: "Alice",
  player2: "Bob",
  "won lost": "1 12", // Valid only when enclosed with quote marks
  printID: function () {
    console.log(this.id);
```

```
  },
};
```

- Object literals can contain other objects:

```
let paramModule = {
  /* parameter literal */
  Param: {
    minGames: 1,
    maxGames: 100,
    maxGameLength: 30,
  },
  printParams: function () {
    console.table(this.Param);
  },
};
```

4.5.3 Design Pattern I: Basic constructor

- In JS, OOP is achieved using functions, constructors and `this`:

```
function Game(id) {
  this.id = id;
  this.totalPoints = 0;
  this.winner = null;
  this.difficulty = "easy";

  this.getID = function () {
      return this.id;
  };
  this.setID = function (id) {
      this.id = id;
  };
}
```

- Objects are initialized using `new`:

```
var g1 = new Game(1);
g1.getID();
```

```
g1.setID(2);
var g2 = new Game(3);

//ES6: object destructuring allows us to extract several object
//     properties at once instead of one-by-one
var { totalP, winner, diff } = g1;
//ES6: template literals to make string concatenations more readable
console.log(
  `This game reached ${totalP} points, was won by ${winner} and had difficulty ${d
);
```

- JS runtime won't alert you in case you forget the `new` keyword when creating an object, however it is very important that you use it. If you create an object without `new`, its `this` keyword will affect the global object (`window` if the code is run inside of a browser).
- With JS, you can add new properties and methods to an object instance after creation:

```
function Game(id) {
  this.id = id;
  this.getID = function () {
    return this.id;
  };
  this.setID = function (id) {
        this.id = id;
  };
}

var g1 = new Game("1");
g1.player1 = "Alice";

var g2 = new Game("2");
g2.player1 = "Bob";

g1.printPlayer = function () {
  console.log(this.player1);
}; //we add a method on the fly!
g1.printPlayer(); //prints out "Alice"

```

```
22 g2.printPlayer(); //TypeError: g2.printPlayer is not a function
23
24 g1.hasOwnProperty("printPlayer"); //true
25 g2.hasOwnProperty("printPlayer"); //false
26
27 g1.toString(); //"[object Object]" (we never defined toString() )
28 // IMPORTANT: Objects come with default methods, as illustrated with
29 // the toPrint() function.
```

1. Summary
   - Advantages:
     - Easy to use
   - Disadvantages:
     (a) Not obvious how to use inheritance
     (b) Objects do not share functions
     (c) There are no private members

4.5.4 Design Pattern II: Prototype-based constructor

- In JS, objects come with default methods. These methods are there because of **prototype chaining**.
- Objects have a secret pointer to another object-the object's prototype. The properties of the constructor's prototype are also accessible in the new object.
- You can manually "walk up" the prototype chain of an object `obj` by calling `obj.__proto__`. However, JS runtime usually does that for you.

1. Summary
   - Advantages:
     - Inheritance is easy to achieve
     - Objects share functions
   - Disadvantages:
     - No public/private distinction