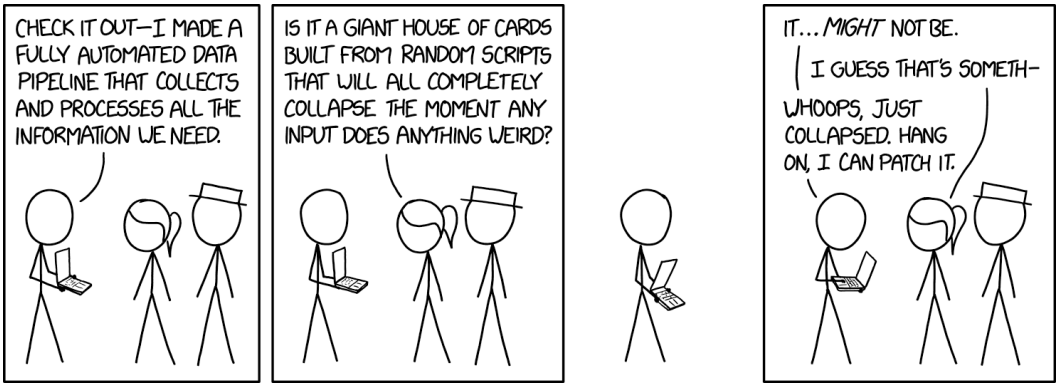
Web & Database Technology — Summary

Dany Sluijk

November 2018



Abstract

This document contains a summary of the Web & Database Technology course given in the first year of Computer Science and Engineering. This is *not* a definitive guide, and might contain errors. Please send an email to "dany@atlasdev.nl". This summary is distributed under the MIT license.

Contents

1 HTTP

HTTP is one of the most important protocols of the internet. It's a human readable and text based, making it easy to use. This summary is talking about version 1.1. Any other version is out of scope.

1.1 Protocol

An example HTTP request looks like the following:

```
POST /say/hello HTTP/1.1
Host: example.com
Content-Length: 12
Content-Type: text/plain
Accept-Encoding: gzip, deflate
Accept-Charset: utf-8
Accept: text/html

Hello world!
```

The first line is the **request line**. It contains the **method** (*POST*) and the **path** (*/say/hello*).

Moving on we see the first header. Headers are key-value objects which indicate metadata about the request. The first one is *Host*, which gives which domain you are requesting from. Together with the request line gives a valid HTTP/1.1 request.

The following five lines are other headers. More on these headers can be found within the Headers subsection.

The next line is just empty. This indicates that the headers have come to an end.

After that line the payload starts. The length of this indicated by the *Content-Length* header. Please note that not all requests have a payload. For example, all *GET* Methods don't.

Ending this with a double newline ends the request. Then the response will be send by the server.

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
```

```
<html>
  <head>
    <title>Hi there!</title>
  </head>
  <body>
    Hello back!
  </body>
</html>
```

This is an example response. As you can see this looks pretty similar to the request.

The first line is the **response line**. This contains the status code off the request. More about the status codes can be found in the Status codes subsection.

The next four lines are response headers, similar to the request headers. They also contain metadata about the request.

After the headers comes a blank line, just like in the request.

Following this is the payload. Contradictory to the request statement this is generally required. There are some exceptions, for example the *HEAD* method.

1.2 Methods

The way to differentiate between actions are methods. Different methods have different meanings. Methods which have no significant side effect other than retrieval are called **safe** methods. Methods which which do have side effects, but should not repeat the side effect when the request is repeated are **Idempotent**.

Method	Description	Safe	Idempotent
GET	Fetch the specified resource	T	T
HEAD	Returns the header of a GET request	T	T
POST	Create a new resource	F	F
PUT	Save the body of the request on the server	F	T
TRACE	Can be used to trace the request	T	T
OPTIONS	Determine what methods the resource supports	T	T
DELETE	Delete a resource from the server	T	F
PATCH	Update a resource on the server	F	F

Table 1: HTTP Methods

1.3 Headers

Headers are an important part of HTTP request. They give metadata about the request and response. This section talks about the most common headers used.

1.3.1 Content-Type

This is the MIME of the body. It is used to give an indication how to parse the body.

1.3.2 Content-Length

This indicates the length of the body. It's a simple integer, in bytes. It is used to make sure all of the message has come trough.

1.3.3 Content-Language

It's a lesser used header which specifies the natural language used in the body.

1.3.4 Content-Encoding

This is used when the server made use of compression to send the body. It indicates how the client should parse the body.

1.3.5 Content-Range

This is used when the body is split up in pieces. It specifies which range of the content is inside of the body. The most common use case is streaming of video.

1.3.6 Content-MD5

This contains the MD5 checksum of the body. It's used to verify that the body is received correctly. Note that this is not a measure against malicious modifications. This is a deprecated header, and rarely used anymore.

1.3.7 Last-Modified

This is the date the resource was updated the latest.

1.3.8 Expires

This gives the maximum amount of time the request may be cached by the client or any cache servers.

1.3.9 Connection & Upgrade

This is used for WebSocket upgrades.

1.4 Status codes

Status codes are used to indicate the result of a request. There are five different types of status codes.

Code	Meaning
1xx	Informational, mostly used as a intermediate code
2xx	Success, the request was processed successfully
3xx	Redirection, the request is somewhere else
4xx	Client made an error in the request
5xx	Server made an error in the request

Table 2: HTTP status codes

The most common status codes are as follows

Code	Name	Meaning
101	Switching Protocols	The server is a new protocol
200	OK	The request was successful.
301	Moved Permanently	The requested resource was permanently moved
302	Found	The resource was found, but was temporally moved
400	Bad Request	There was an error in the request
401	Unauthorized	Authentication was required, but not provided
403	Forbidden	Access to the resource has been denied
404	Not Found	The resource was not found
405	Method Not Allowed	The given request method is not allowed
418	I'm a teapot	The server is a teapot
500	Internal Server error	A general error occurred on the server
502	Bad Gateway	Invalid response from a upstream server

Table 3: most common HTTP status codes

2 URLs

URLs are a important part of the internet. The acronym stands for **Uniform Resource Locator**. As you can see it gives an indication where a certain resource can be found.

2.1 format

An URL is formatted as the following:

Figure 1: Format of an URL.

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

Let's talk about this, part for part.

2.1.1 Scheme

The **scheme** indicates the protocol to use for the URL. It will tell the operating system which program can process it. The most common schemes used are *http://*, *https://* and *mailto://*. This format explained is for HTTP and HTTPS. An other scheme can use a different format.

2.1.2 Auth

This format is deprecated, but still used and accepted wildly. It can provide a username and password for authentication. Most of the times it is *base64* encoded. The basic format is as following:

```
username:password
```

2.1.3 Host

This is the domain name most of the time. It refers to the location of the server this is connected to. If it's a domain name it'll get resolved by the **DNS** first.

2.1.4 Port

Here you can override the TCP/IP port used to create a connection. Normally this is 80 for HTTP, and 443 for HTTPS.

2.1.5 Path

This is the path where the resource is located. THis indicates the location of the resource inside of the server. An example is:

```
/this/is/my/path.html
```

2.1.6 Params

Here you can give more parameters to the url. This is uncommon practice to use. I recommend to use the *query* part instead.

2.1.7 Query

This is used to pass parameters via the URL. These are simple key value parameters.

```
?foo=bar&hello=hi
```

2.1.8 Frag

A frag is a location inside of a document. It is most often used for referring to a specific chapter in a HTML document (an anchor).

2.2 Punycode

URLs were intended to only be used with ASCII characters. This was fine in the beginning, as the internet was entirely in english. In the modern day this is no longer the case, it's being used all over the world. With other languages come other characters, characters which are not in ASCII. To fix this issue punycode was invented. It's a type of encoding to encode other characters with only ASCII. For example:

```
xn{pple-43d.com
```

gives:

```
apple.com
```

2.2.1 Phishing

Punycode does make the browser vulnerable to a specific type of homograph attacks. As you can see in last example the punycode renders as apple.com. It is not actually apple.com, which can cause confusion. You can even get SSL certificates for these domains, making is way more convincing.