Web Services, HTTP and REST

Reading / References

- Restful Web Services[1], chapters 1-4, especially chapter 4. The appendices contain information about HTTP.
- HTTP on MDN[2]
- Browser Networking[3] free online book, optional
- HTTP specification (more a reference than a read)[4]
- HTTP the definitive guide[5] reference
- Fun read: Hyper Text Coffee Pot Control Protocol[6]

Practice questions on the reading

- What kind of information can we send over the HTTP protocol?
- What are the basic parts of a HTTP request and an HTTP response?
- List 5 important HTTP Response/Status Codes and their meaning.
- List 8 important HTTP headers and their meaning. At least 3 headers should be request headers, and at least 3 should be response headers.
- Does the HTTP protocol specify what can go into the body part of requests or responses?

Notes

We will discuss Web Services and related concepts in this section. We start with some key terminology:

Web Service A service provided over the internet via HTTP, and usually targetted at automatic consumption via other computers or programs. In that way it differs from a personal or company website.

Web API API in general stands for Application Program Interface. In this particular instance it describes how the Web Service expects clients to interact with it. Since its primary clients are computers themselves, the API needs to be formally described.

HTTP A simple protocol for transfer of data over an internet. HTTP establishes a simple way to describe resources and how to request and receive them, and we will discuss this extensively in the future.

[1] https://www.safaribooksonline.com/library/view/restful-web-services/9780596529260/
[2] https://developer.mozilla.org/en-US/docs/Web/HTTP
[3] https://hpbn.co/
[4] http://www.w3.org/Protocols/rfc2616/rfc2616.html
[5] https://www.safaribooksonline.com/library/view/http-the-definitive/1565925092/
[6] https://en.wikipedia.org/wiki/Hyper_Text_Coffee_Pot_Control_Protocol

REST A particular way of describing and serving the resources via HTTP that uses some of the strengths of HTTP. Fairly accepted nowadays as a standard way to build a web service. Almost every programming language offers a framework that implements REST, and many web services follow a RESTful API.

HTTP

HTTP stands for HyperText Transfer Protocol. It is what is known as an "Application Layer Networking protocol". This means that it does not concern itself with how network packets will traverse the internet, but instead aims at a higher-level description of the information in those packets. It is the primary mode of communication between web browsers and web servers. Every time you request a webpage, HTTP is the protocol that arranges for that webpage to find its way to you.

There are some key characteristics that determine the behavior of HTTP:

addressable Every resource you may want to access has a corresponding Uniform Resource Identifier (URI), what we all affectionately know as a URL. A simple, clear, uniquely identifiable access point for the resource. URIs have the general form (example taken from Wikipedia):

```
scheme:[//authority]path[?query][#fragment]
https://www.example.com:123/forum/questions/?tag=networking&order=newest
```

where the first part is the protocol used, followed by the internet name for the server we are trying to access, typically called the **hostname**, optionally including a **port** number (123 in the example above). This is then followed by the **path** to the resource we want to reach. A question mark indicates the beginning of a **query** component which contains extra parameters for the request.

stateless The protocol is "stateless". In other words, each request that the client sends to the server has no memory of prior requests and replies.

This is an important feature of the protocol, and something that actually lent to its popularity. It makes the implementation of it on both the server and client side easier, and makes the overall protocol simpler, as neither side needs to maintain any information from previous requests. Other protocols had been proposed around the same time, but HTTP won in the end as the standard, in large part due to its simplicity.

One of the consequences of course is that in situations where we need to maintain the history of what has occured, both browser and server need to agree on a way to do that (e.g. keeping someone "logged in").

client/server The HTTP protocol is characterized by the non-symmetric description of the two parties involved. One party is the *client* and the other is the *server*. The client sends **requests** to the server, and the server sends back **responses**.

session The typical interaction between client and server, called a *session*, would go something like this:

1. Client establishes a network connection with server. This is typically done via TCP/IP and requires some amount of initial setup.
2. Client sends a request packet and waits for the answer.
3. Server receives the packet, then prepares and sends a response.
4. In HTTP 1.1 and later, the client may send further requests and receive responses.
5. When the client has no more requests, they close the connection.
6. The connection may also "time out" after a given amount of time.

Clients and servers specify themselves via their IP addresses (and port numbers). This is taken care of at the TCP/IP layer.

request The client sends an "HTTP request" to the server over the network. That request includes the **request method**, followed by the **resource path** as well as the protocol version, typically HTTP/1.1.

It will be followed by a series of **headers**, that can identify various aspects of the request, like the content type, the host name, the content length, the accepted languages for the reply and so on.

Some request methods allow extra content, which can be found below the headers. Here is an example from the MDN page:

```
POST /contact_form.php HTTP/1.1
Host: developer.mozilla.org
Content-Length: 64
Content-Type: application/x-www-form-urlencoded

name=Joe%20User&request=Send%20me%20one%20of%20your%20catalogue
```

This says that the request uses the POST method and the resource path is /contact_form.php. There are 3 headers, one specifying the host, and two more specifying the details about the content. After that and following an empty line we find the content (name...).

Here are the main request methods. Think of these as "function calls". A web browser can typically only use the first two, but the client of a web service could use more.

GET A GET request asks for some resource and is not meant to cause any changes to the server.

POST A POST request is used when submitting forms for example. It is typically expected to be used for updating some server information.

PUT Used for changing information of some server resource. Often performed via a POST instead, but that violates the REST principles we will discuss later.

HEAD The reply will contain just the header information, without any content body. It is useful as an early interaction step with a server in order for the client to find out what kind of headers to provide along with the "true" request.

DELETE Used for deleting server resources (if allowed).

response Server responses have a special first line containing the protocol followed by the **response status code**. This is followed by response headers, and finally the content body of the response. Here is an example:

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

There are many different request and response headers, look at the documentation for more information.

Here are some typical response status codes, there are many more:

200 *OK*. The resource was available and is returned. In general, all codes in the 2xx range indicate various kinds of successful operations.

301 *Moved Permanently*. The resource has been moved to a different location. A `Location` header gives the new location. In general, all codes in the 3xx range indicate some sort of redirection.

304 *Not Modified*. The resource hasn't been modified since the last time we asked for it. Our cached version will do just fine.

404 *Not Found*. The requested resource cannot be found. Typically the result of typos in the request. In general, all codes in the 4xx range indicate some sort of client error.

500 *Internal Server Error*. Usually indicates problems with the server's configuration/availability. In general, all codes in the 5xx range indicate some sort of server error.

You typically don't have to directly create these requests and responses as text, there are libraries that do that for you and allow you to talk about these responses on the level of objects.

Activity

In this activity one student acts as the server, while other students act as clients making HTTP requests of the server. The server is required to send a response for each request. Here are the instructions for the server:

- You are providing a simple arithmetic service. It accepts two resource addresses, `/add` and `/subtract`.
- You used to also support `/minus`, but since the creation of `/subtract` you want to encourage all your users to transition to it, by providing a redirect.

- There are no other resources that you are offering. You should be returning a suitable status code if someone requests another resource than the above.
- You are only accepting POST requests. If a request is not a POST request, you should respond in a suitable way.
- You are requiring all requests to provide the content's length in the Content–Length header. If that header is not present, you should return a suitable status code.
- The body of the request should contain the two numbers to add or subtract, separated by a space. Any other form of the body text should result in a bad request.
- Successful requests should receive the result of the arithmetic computation in the body of the response. You will also need to correct the Content–Length header value depending on the size of the result.