Datanet Assignment 1

Christian Enevoldsen
DIKU
Vesterbrogade 20c 1tv
1620, kbh v
+45 5376816
c.enevoldsen@gmail.com

ABSTRACT
In this paper, we will describe a simple implementation of a http client which can do get request and store them in files.

1. INTRODUCTION
Every day we use the internet for several things. One may watch youtube videos, read the newspaper, chat, socialise etc. This is mainly done through a web browser which tightly communicates with a webserver in such a way that the webbrowser converts actions into requests which a server will handle and respond to. In this report we focus on the abstraction of a simple http client which can be used to do GET requests on the servers.

2. DESIGN
The http GET client is written in Javascript and compiled using io.js which is a branch of the popular node.js. It runs with the V8 engine and compiles javascript into machine code for faster computations. The client uses the following built-in libraries

- net (for creating socket connections)
- fs (for writing files)

3rd party libraries

- util-extend (eases the process of extending JSON objects)

2.1 URL PARSING
Url parsing is done in src/url.js. This module has one method: parse which main job is to convert a URL into a dictionary like type which includes the protocol, host, hostname, port, path, pathName, query and finally the entire HREF.

The parsing is done with a greedy like algorithm such that we start by finding the protocol and then removes that from the url. Then we split by a / to get the host and host name and then remove that from the url so we by now only have the path URI left. For instance /index.html. One could argue that it would be easier with regular expression, however I find that it is harder to read and much slower.

2.2. THE CLIENT
The client is factored into its own module which you can find in src/http-client.js — for now the only method you will find useful is the get request method. It takes as parameters an URL, options and a callback. The options may be omitted, but if one might want to set a header you do so in the following manner:

The last ‘end’ takes the result and parses into an object (headers and the body). After the parsing we call the callback with the final result.

```
var options = {
  headers: { Connection: 'close' }
};


client.get(url, options, function(err, req, res) {
```

The get request calls the internal request method and specifies that it should do a get request. This is done so that it is easily extendable for other request types.

The url is first parsed into an object such that we easily can extract what is needed. For instance the host. After that the request headers are built. The default headers would look like this after it is built

GET / HTTP/1.1

User-agent: simple-get

Host: www.host.dk

Connection: close (optional)

Next we use the net library to create a connection to the host on a specific port (default port is 80)

Then we define some callbacks that will be used to receive data and do the request

client.on('data', function(data) {…

client.on('connect', function() {…

client.on('end', function() {…

The first ‘data’ is called whenever the client receives data. The data is then added to a local response object. We do this because the data is sent in chunks and not as a whole.

The second ‘connect’ is called once the connection is established. The only thing that is done here is to call the client.end method. Which will take the request, send it and finally close the connection

2.3. Measurement

For simplicity and because measurement should not be a necessary in a http-client, it's refactored into a submodule. The measurement works in such way that you create an instance of the

Measure class, var m = new Measure(), then you call m.begin() before the request and m.end() when the request is done. Now you can get the elapsed time by the Measure.elapsed property. To reset just call begin() again.

3. LIMITATIONS

The client can only handle text based documents for now, since the data received is represented using .toString() method. This results in other file formats to be stored incorrectly and therefore the md5 hash will differ. The toString() also has the limitation that it doesn't encode the text quite well. For instance downloading the google.dk page and storing it in an html file and displaying it in a browser results in some characters being nonsense.

4. TESTS

The following tests has been done.

File:

http://ftp.au.debian.org/debian/pool/main/0/0ad/0ad_0.0.17-1_amd64.deb

Using the client: 5 times

Average 6 seconds and 447.373 ms

Using chrome: 5 times

Average 6 seconds and 432.212 ms

It's hard to tell if there's a difference because the connection might have been in favor of the client in when the tests was done and vice versa. Conclusion is that there's no big difference.