Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Department of Computer Science

REPORT

Laboratory no. 5

HTTP protocol: mean of distributed data transmission

Author:
Andrei Capastru

Lecturer:
Dumitru Ciorbă

January 21, 2016

Contents

List of Figures

Listings

1 Objective

The goal of the laboratory study lies on the HTTP protocol in the context of data distribution and the use of HTTP methods in implementing the interaction between the client and the server applications.

The primary objectives:

— Creation of a server with concurent processing of HTTP requests

— Implement GET/POST methods for processing the requests

2 Theoretical background

2.1 HTTP

The Hypertext Transfer Protocol (HTTP) [1] is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a clients request message. Upon receiving the request, the server sends back a status line, such as HTTP/1.1 200 OK, and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.

2.2 Request methods

HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

The HTTP/1.0 specification defined the GET, POST and HEAD methods and the HTTP/1.1 specification added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. By being specified in these documents their semantics are well known and can be depended upon. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined and this allows for future methods to be specified without breaking existing infrastructure. We focused on 2 particular methods:

1. `GET` - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.) The W3C has published guidance principles on this distinction, saying, Web

application design should be informed by the above principles, but also by the relevant limitations.

2. POST - The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

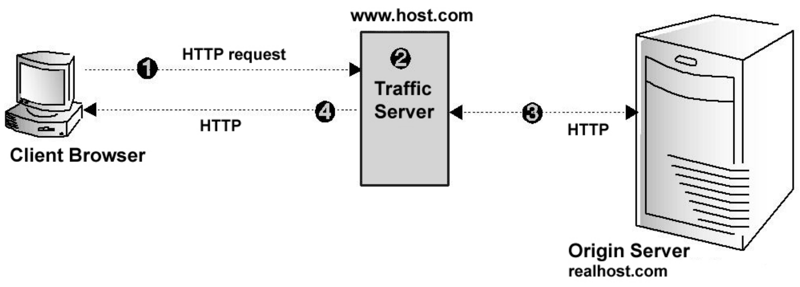

Figure 1: HTTP Request Illustration

3 Solution description

As server application was used Node.js (JavaScript environment). [2]Modules used in this application:

- `http` — zero-configuration command-line http server. It is powerful enough for production usage, but it's simple and hackable enough to be used for testing, local development, and learning.
- `httpdispatcher` —A simple class allows developer to have a clear dispatcher for dynamic pages and static resources.
- `underscore` — JavaScript's functional programming helper library.
- `url` — The core url packaged standalone for use with Browserify.

3.1 Loading HTTP module

Node.js is shipped with several core modules out of the box, which we will be using to set up our own http server. The http module makes it simple to create an http server via its simple but powerful api.

```
var http = require('http');
function handleRequest(request , response){
    response.end('It Works!! Path Hit: ' + request.url);
}
```

Listing 1: Loading HTTP Module

3.1.1 Loading the server

```
var server = http.createServer(handleReq);
server.listen(PORT, function(){
    console.log("Server on: http://localhost:", PORT);
});
```

Listing 2: Server load

3.1.2 Dispatching

Now, that the server is up and running it can receive information. That information comes ”patched” and it needs to be dispatched to work on it. For this purpose is used the `httpdispatcher` that takes response and dispatches it in convenient data. It can do this for POST and GET HTTP methods. The POST dispatching is done in the following function:

```
dispatcher.setStatic('resources');

dispatcher.onPost("/", function(request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Got Post Data' + request.body);
  console.log('Data received' + request.body)
  warehouse.push.apply(warehouse, JSON.parse(request.body));
});
```

The following GET routes are set:

— `http://localhost:1234/data?id=1`
— `http://localhost:1234/alldata`
— `http://localhost:1234/alldata?offset=1limit=4`
You can access them using a usual web browser as it use get method for accessing websites.

3.2 Workflow

The following steps are done during application testing:

1. Run Server using `node server.js`
2. Run a Post request using `node post.js` (or `post1.js` for diversity)
3. Access data via get request using Postman or a Webrowser.

4 Conclusion

During elaboration of this laboratory work I got to know more about HTTP, its methods and what means a REST API.

I also got to know some more Node.js modules described above and use them in creating simple web servers and applications.

References

[1] Compaq H. Frystyk W3C/MIT L. Masinter Xerox P. Leach Microsoft T. Berners-Lee Compaq/W3C J. Mogul. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: `https://tools.ietf.org/html/rfc2616`.

[2] Node.js. *npm*. 2012-2015. URL: `https://www.npmjs.com`.