Full Stack Web Development

WS2019 - BSA5 Ausgewählte Kapitel

Alija Sabic
sabic@technikum-wien.at

Smart Homes and Assistive Technologies October 28, 2019

Outline

1. Node.js

Section

1 Node.js

1.1 History

1.2 Installation

1.3 Applications

1.4 HTTP Module

Node.js - Ryan Dahl [2, 3]

- Born and raised in San Diego, California.
- Studied mathematics at the UC San Diego.
- Dropped out of the PhD program and bought one-way ticket to South America, where he found a job as web developer.
- Started the work on Node.js in 2009.
- Stepped away from active development on Node.js in 2012.
- Started the work on Deno in 2018.

Figure 1: Ryan Dahl [1]

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js

- JavaScript runtime
- Cross-platform
- Single-threaded
- Asynchronous and event driven
- Fast (Google's JavaScript Engine V8)

Figure 2: Node.js Logo [4]

Alija Sabic

Node.js - History I [2]

- 2009: In January, Ryan Dahl starts work on a JavaScript runtime named Node.js based on Google's V8.
- 2009: In September, Mark Schuelter starts work on a Node Package Manager, further known as npm.
- 2009: In November, Ryan Dahl presents Node.js at the JSConf EU in Berlin [5].
- 2011: v0.6.0 is released adding support for Windows OSs.
- 2011: v0.6.3 ships npm with installation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History I [2]

- 2009: In January, Ryan Dahl starts work on a JavaScript runtime named Node.js based on Google's V8.
- 2009: In September, Mark Schuelter starts work on a Node Package Manager, further known as `npm`.
- 2009: In November, Ryan Dahl presents Node.js at the JSConf EU in Berlin [5].
- 2011: v0.6.0 is released adding support for Windows OSs.
- 2011: v0.6.3 ships `npm` with installation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History I [2]

- 2009: In January, Ryan Dahl starts work on a JavaScript runtime named Node.js based on Google's V8.
- 2009: In September, Mark Schuelter starts work on a Node Package Manager, further known as `npm`.
- 2009: In November, Ryan Dahl presents Node.js at the JSConf EU in Berlin [5].
- 2011: v0.6.0 is released adding support for Windows OSs.
- 2011: v0.6.3 ships `npm` with installation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History I [2]

- 2009: In January, Ryan Dahl starts work on a JavaScript runtime named Node.js based on Google's V8.
- 2009: In September, Mark Schuelter starts work on a Node Package Manager, further known as `npm`.
- 2009: In November, Ryan Dahl presents Node.js at the JSConf EU in Berlin [5].
- 2011: v0.6.0 is released adding support for Windows OSs.
- 2011: v0.6.3 ships `npm` with installation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History I [2]

- 2009: In January, Ryan Dahl starts work on a JavaScript runtime named Node.js based on Google's V8.
- 2009: In September, Mark Schuelter starts work on a Node Package Manager, further known as `npm`.
- 2009: In November, Ryan Dahl presents Node.js at the JSConf EU in Berlin [5].
- 2011: v0.6.0 is released adding support for Windows OSs.
- 2011: v0.6.3 ships `npm` with installation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History II [2]

- 2014: Community fork of Node.js, called io.js.
- 2015: In February, the Node.js Foundation is established, supervising future development of io.js.
- 2015: In June, both projects, Node.js and io.js, join the Node.js Foundation. Future development is guided by a *Technical Steering Committee* (TSC).
- 2015: In September, a joined version (v4.0.0) is released. Regular release cycles. Even version numbers are *Long Term Stable* (LTS) releases.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History II [2]

- 2014: Community fork of Node.js, called io.js.
- 2015: In February, the Node.js Foundation is established, supervising future development of io.js.
- 2015: In June, both projects, Node.js and io.js, join the Node.js Foundation. Future development is guided by a *Technical Steering Committee* (TSC).
- 2015: In September, a joined version (v4.0.0) is released. Regular release cycles. Even version numbers are *Long Term Stable* (LTS) releases.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History II [2]

- 2014: Community fork of Node.js, called io.js.
- 2015: In February, the Node.js Foundation is established, supervising future development of io.js.
- 2015: In June, both projects, Node.js and io.js, join the Node.js Foundation. Future development is guided by a *Technical Steering Committee* (TSC).
- 2015: In September, a joined version (v4.0.0) is released. Regular release cycles. Even version numbers are *Long Term Stable* (LTS) releases.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - History II [2]

- 2014: Community fork of Node.js, called io.js.
- 2015: In February, the Node.js Foundation is established, supervising future development of io.js.
- 2015: In June, both projects, Node.js and io.js, join the Node.js Foundation. Future development is guided by a *Technical Steering Committee* (TSC).
- 2015: In September, a joined version (v4.0.0) is released. Regular release cycles. Even version numbers are *Long Term Stable* (LTS) releases.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module

Node.js - Versions

Since the establishment of the Node.js Foundation, Node.js is release regularly.

Release	Codename	Initial Release	End of Life
4.x LTS	Argon	08.09.2015	April 2018
5.x		29.10.2015	June 2016
6.x LTS	Boron	26.04.2016	April 2019
7.x		25.10.2016	June 2017
8.x LTS	Carbon	30.05.2017	December 2019
9.x		01.10.2017	June 2018
10.x LTS	Dubnium	24.04.2018	April 2021
11.x		23.10.2018	June 2019
12.x LTS		23.04.2019	April 2022

Table 1: Node.js release schedule.

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation

The simplest way to install Node.js is install it with a package manager or to download the current version from the Node.js download page.

Node.js provides installers, prebuild binaries for different operating systems and architectures, together with other options like Docker container (cf. Figure 3).

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation

Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.zip)	32-bit	64-bit	
macOS Installer (.pkg)	64-bit		
macOS Binary (.tar.gz)	64-bit		
Linux Binaries (x64)	64-bit		
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v10.16.3.tar.gz		

Additional Platforms

SmartOS Binaries	64-bit
Docker Image	Official Node.js Docker Image
Linux on Power Systems	64-bit
Linux on System z	64-bit
AIX on Power Systems	64-bit

Figure 3: Node.js Downloads [4]

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Linux

To install the Node.js binary in the `opt/` directory, run following commands.

```
~ $ cd ~/Downloads
~/Downloads $ wget \
> https://nodejs.org/dist/v10.16.3/node-v10.16.3-linux-x64.tar.xz

--2019-10-14 00:38:55--  https://nodejs.org/...
Resolving nodejs.org (nodejs.org)... 104.20....
...

~/Downloads $ cd opt/
/opt $ sudo tar xvf ~/node-v10.16.3-linux-x64.tar.xz

[sudo] password for <username>:
node-v10.16.3-linux-x64/
...

/opt $ sudo ln -s node-v10.16.3-linux-x64 nodejs
/opt $ PATH=$PATH:/opt/nodejs/bin
/opt $ export PATH
```

Listing 1: Install Node.js binary

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Linux

The command to install Node.js from a package repository depends on the package manager of your distribution. On Ubuntu, run following commands.

```
~ $ sudo apt-get install nodejs

Reading package list... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libc-ares2 libhttp-parser2.7.1 libuv1 nodejs-doc
...
```

Listing 2: Install Node.js with package manager

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Linux

You can add a package repository to have always the latest version when installing with the package manager.

```
~ $ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
~ $ sudo apt-get install -y nodejs

## Installing the NodeSource Node.js 10.x repo...

## Populating apt-get cache...
...
```

Listing 3: Install Node.js with package manager

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Windows

To install the Node.js binary in the `C:` directory, download the provided archive and extract in to the drive.

You can now start Node.js from a terminal inside of the directory.

```
C:\node-v10.16.3-win-x64\> node.exe -v

v10.16.3
```

Listing 4: Test Node.js version

To access the Node.js binary from every location, you have to add the folder containing the binary to the PATH environment variable.

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Windows

To install the Node.js binary and setup environment variables automatically, use the provided MSI installer.

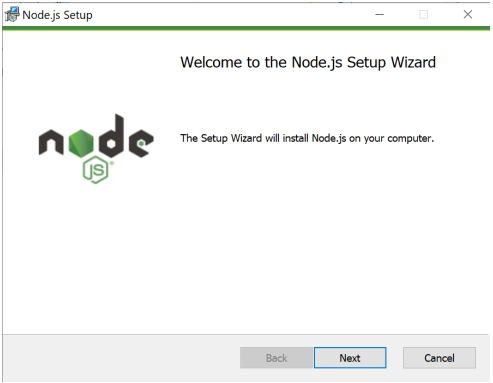

Figure 4: Start Node.js MSI installer

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Windows

After the installation is complete, you will find several programs in the Node.js start menu entry.

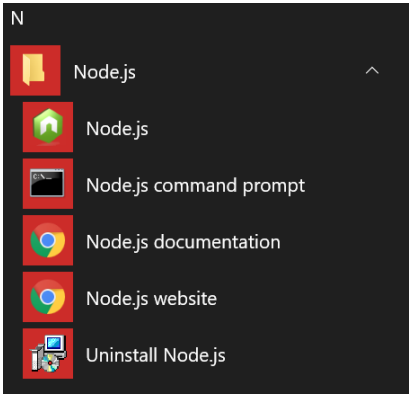

Figure 5: Start menu entry of Node.js

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Windows

The first entry, Node.js, starts Node.js in REPL mode.

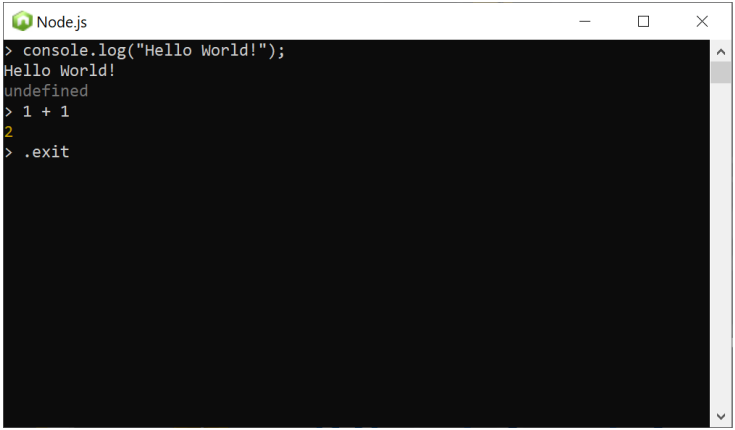

Figure 6: Node.js REPL mode

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on Windows

The second entry, Node.js command prompt, opens a terminal window with a set up environment.

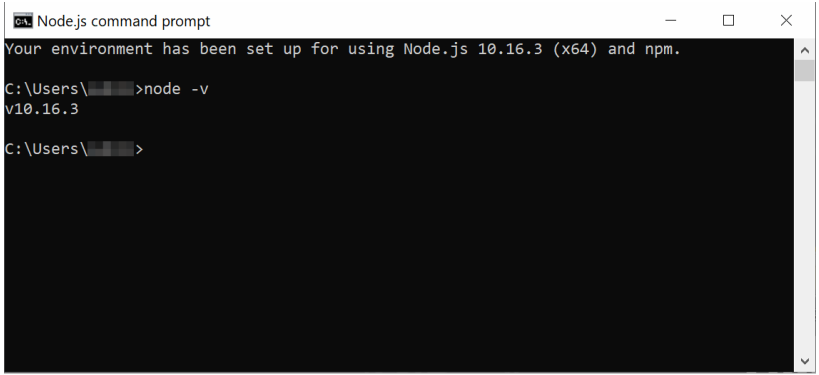

Figure 7: Node.js command prompt

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Installation on macOS

To install the Node.js binary on macOS, run following commands.

```
~ $ wget https://nodejs.org/dist/v10.16.3/node-v10.16.3-darwin-x64.tar.gz

--2019-10-14 00:38:55--  https://nodejs.org/...
Resolving nodejs.org (nodejs.org)... 104.20....
...

~ $ cd tar xvzf node-v10.16.3-darwin-x64.tar.xz

node-v10.16.3-linux-x64/
...

~ $ cd node-v10.16.3-darwin-x64
~/node-v10.16.3-darwin-x64 $ bin/node -v

v10.16.3
```

Listing 5: Install Node.js binary

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Building from Source

Since Node.js is an open-source project, the source files can be downloaded and used to build an individual version of Node.js.

The following commands outline how to build Node.js from source on Ubuntu.

```
~ $ cd Downloads
~/Downloads $ wget https://nodejs.org/dist/v10.16.3/node-v10.16.3.tar.gz
~/Downloads $ tar xfz node-v10.16.3.tar.gz
~/Downloads $ cd node-v10.16.3

# List available options
~/Downloads/node-v10.16.3 $ ./configure --help

# Example: w/o npm, w/o crypto, https, etc.
~/Downloads/node-v10.16.3 $ ./configure --without-npm --without-ssl
```

Listing 6: Building Node.js from source

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Building from Source

To configure, build and install Node.js with typical defaults, run following commands.

```
# Configure
~/Downloads/node-v10.16.3 $ ./configure --prefix=/opt/node/node-v10.16.3

...

# Build
~/Downloads/node-v10.16.3 $ make -j4

...

# Install
~/Downloads/node-v10.16.3 $ sudo make install

...
```

Listing 7: Building Node.js from source

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Node Version Manager

If you need to run or test your scripts with different node versions, you can download the *Node Version Manager* (`nvm`) from the GitHub repository https://github.com/nvm-sh/nvm.

```
~ $ wget -qO- \
> https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.0/install.sh |\
> bash

=> Downloading nvm as script to '/home/<user>/.bashrc'
=> Appending bash_completion source string to /home/<user>/.bashrc
=> Close and reopen your terminal to start using nvm or run ...
...
```

Listing 8: `nvm` Installation

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Node Version Manager

After the installation is finished, you can use `nvm` to install and use different versions of Node.js.

```
# Print help
~ $ nvm --help

# List available node versions
~ $ nvm ls-remote
~ $ nvm ls

# Install current LTS version
~ $ nvm install --lts node

Downloading and installing node v10.16.3
...

# Use current LTS version
~ $ nvm use --lts node

Now using node v10.16.3 (npm v6.9.0)
```

Listing 9: `nvm` Usage Examples

Alija Sabic
Web Development

Node.js
History
Installation
Linux, Windows, macOS
Custom Build
Node Version Manager
Applications
HTTP Module

Node.js - Applications

After successfull installation, you should have following programs on your system.

- `node`
- `npm`
- `npx`

Node.js can be started in different modes.

```
# Read-Eval-Print-Loop (REPL)
~ $ node
# Run
~ $ node script.js
# Debug
~ $ node inspect script.js
```

Listing 10: Node.js - Modes

Alija Sabic
Web Development

Node.js
History
Installation
Applications
Command Line
Server
HTTP Module

Node.js - Applications

Node.js scripts are used to create different types of applications.

- **Command Line**

  Node.js is used for a variety of utility applications for the terminal, like css compiler and linter.

  - `typescript`
  - `eslint`
  - `vue`
  - `sass`

- **Server**

  Most important application field, however, is building server software.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
Command Line
Server
HTTP Module

Node.js - Command Line

To run a script in Node.js run following command.

```
~ $ node hello.js
Hello World!
~ $
```

Listing 11: Running Node.js scripts

```
console.log("Hello World!");
```

Listing 12: Node.js CLI application

Alija Sabic
Web Development

Node.js
History
Installation
Applications
Command Line
Server
HTTP Module

Node.js - Server

The next listing show a “Hello World” server application.

```
1  const http = require("http");
2
3  const server = http.createServer((req, res) => {
4    res.statusCode = 200;
5    res.setHeader("Content-Type", "text/plain");
6    res.end("Hello World\n");
7  });
8
9  const hostname = "127.0.0.1";
10 const port = 3000;
11 server.listen(port, hostname, () => {
12   console.log(`Server running at http://${hostname}:${port}/`);
13 });
```

Listing 13: Node.js server application

Alija Sabic
Web Development

Node.js
History
Installation
Applications
Command Line
Server
HTTP Module

Node.js - Server

Open a web browser an enter the URL displayed in the console.

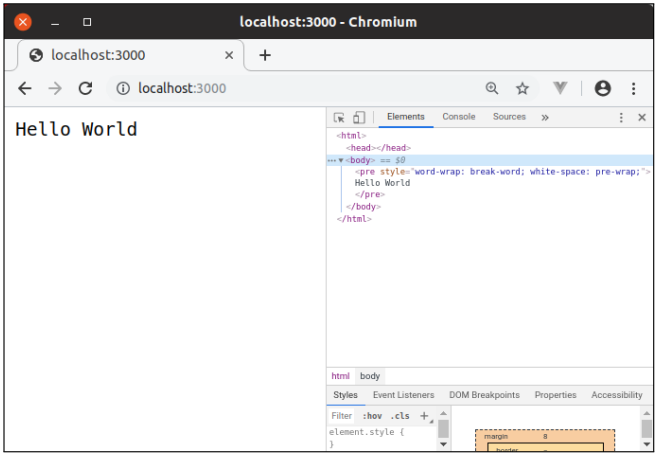

Figure 8: Node.js server application response in browser

Alija Sabic
Web Development

Node.js
History
Installation
Applications
Command Line
Server
HTTP Module

Node.js - HTTP Module

Node.js ships with the `http` module, providing functionality for running HTTP servers and making HTTP requests.

Before we start writing our server application, let's review what's happening behind the scence when we enter an URL in our browser.

- ① - ② DNS lookup
- ③ - ⑤ TCP connection establishment: three-way handshake
- ⑤ - ⑦ HTTP request/response
- ⑦ - ⑧ TCP connection destruction

If there are other files referenced in the HTML file of the HTTP response, the client (browser) will request those as well.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

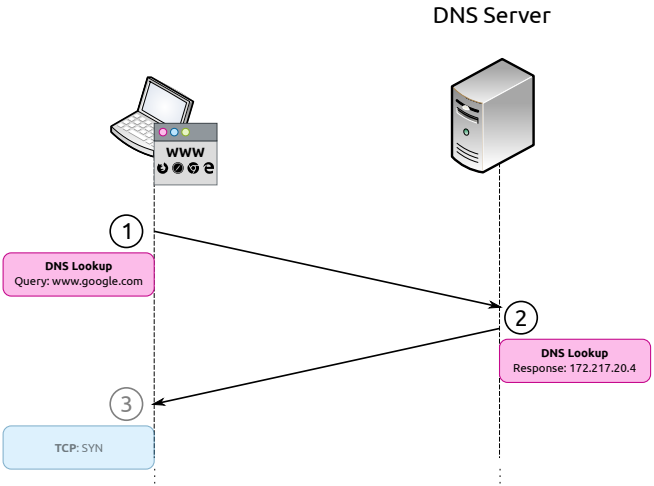

Figure 9: DNS lookup

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

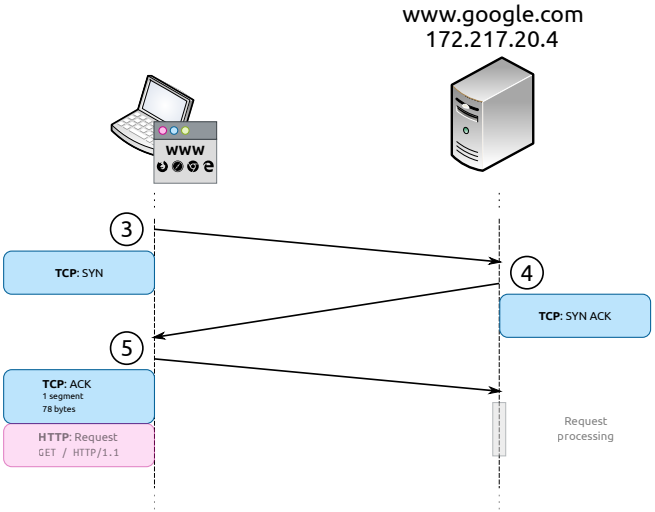

Figure 10: TCP three-way handshake

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

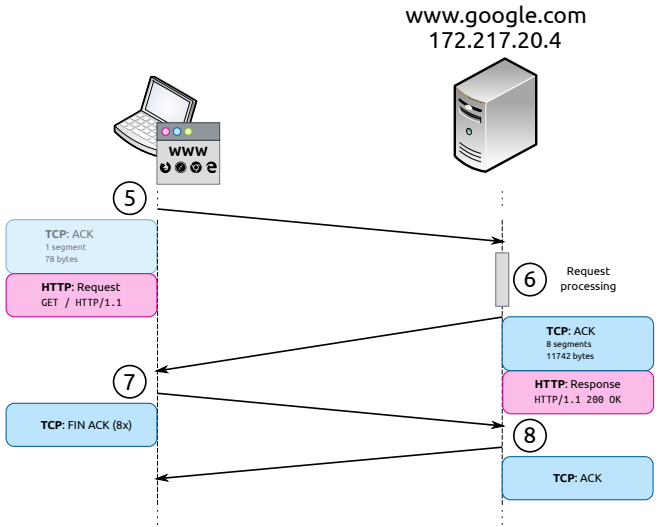

Figure 11: HTTP request/response

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

The following listing show how to instantiate a server object with the HTTP module provided by Node.js and bind it to all network interfaces available on port 8080.

```
1 const http = require("http");
2
3 /* Create a http server */
4 const server = http.createServer();
5
6 /* Bind server to a port */
7 server.listen(8080);
```

Listing 14: Node.js - HTTP Module (`server.js`)

Node.js makes use of the CommonJS module format. The load the functionality provided by the HTTP module, you have to use the `require()` function.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

To start our application, we have to run the created script in the terminal.

```
~ $ node server.js
```

Listing 15: Starting the Node.js server (`server.js`)

The server keeps listening for user request, but currently does not provide any output to the console. If you want to stop Node.js or the server application, hit Ctrl+C.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

When we enter the URL `http://localhost:8080/` in our web browser, it will send a HTTP request to your computer on port 8080.

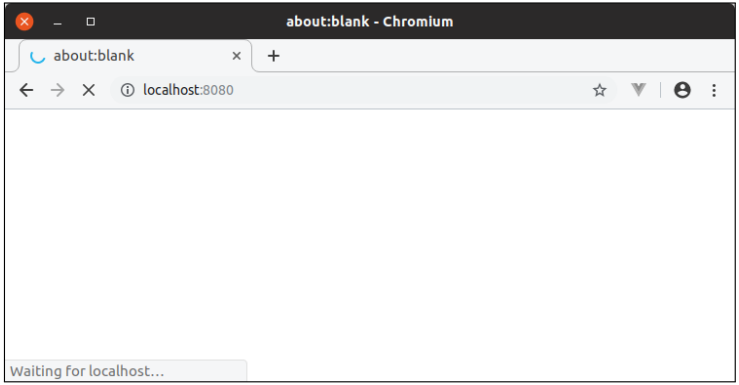

Figure 12: Opening `http://localhost:8080/` in a browser

As you can see, the browser keeps waiting for the localhost to reply.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

In the *Network* tab of the browser devtools, we can see, that the browser has sent a request to `localhost` and the response is still pending.

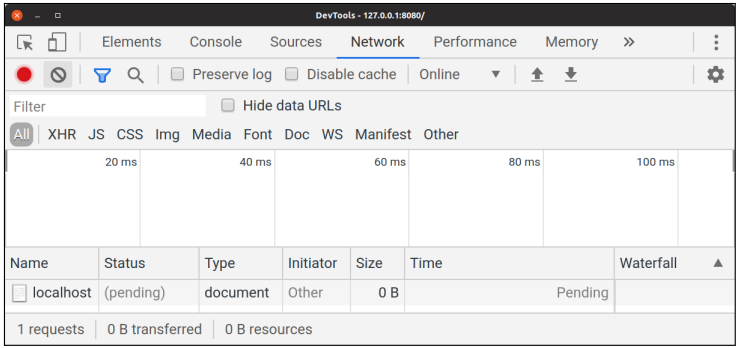

Figure 13: Browser devtools - Network tab

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

To see what packets are being sent, we can use *Wireshark* to capture the packets on the loopback interface.

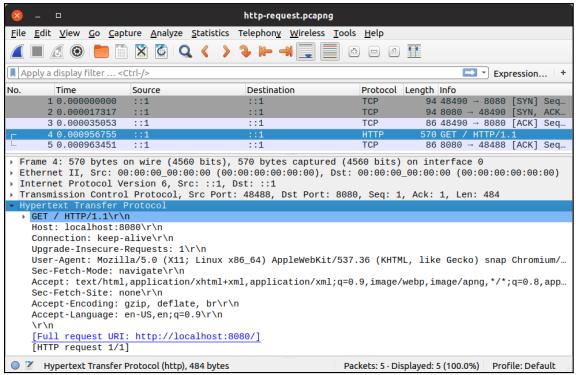

Figure 14: Wireshark capturing HTTP request

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

The client (browser) has sent a request. However, the server did not reply a response to the request of the client.

We have only instantiated a HTTP server so far. We can register a *callback* function, to be invoked when the server instance receives HTTP requests.

```
3 /* HTTP request callback */
4 function requestCallback(request, response) {
5   response.end("Hello Node.js");
6 }
7
8 /* Create a http server */
9 const server = http.createServer(requestCallback);
```

Listing 16: Node.js - HTTP Module (`server.js`): callback registration

When the callback function is invoked, it gets two parameters passed, that is `request` and `response`. We use the `end()` method of the response object, which provides a way to respond on client requests.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

If we restart our server to load the updated script and refresh the current browser window, we will finally receive a response from the server.

Figure 15: Receiving response for `http://localhost:8080/`

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

Since we are not using the function `requestCallback` anywhere else, we can pass an anonymous function to `http.createServer()`.

```
3 /* Create a http server */
4 const server = http.createServer(function(request, response) {
5   response.end("Hello Node.js");
6 });
```

Listing 17: Node.js - HTTP Module (`server.js`): anonymous callback function

`http.createServer()` is only a short form of an instanation of the class `http.Server` and callback registration for `request` events.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

Alternatively, we could have written our script in the following way.

```
1  const http = require("http");
2
3  /* Create a http server */
4  const server = new http.Server();
5
6  /* Register a callback for request events */
7  server.on("request", function(request, response) {
8    response.end("Hello Node.js");
9  });
10
11 /* Bind server to a port */
12 server.listen(8080);
```

Listing 18: Node.js - HTTP Module (`server.js`): alternative approach

We may even use some of the newer features like object destructuring and arrow functions. Pages like node.green provide an overview on the JavaScript features supported by the different versions of Node.js.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

```
1  const { Server } = require("http");
2
3  /* Create a http server */
4  const server = new Server();
5
6  /* Register a callback for request events */
7  server.on("request", (request, response) => {
8    response.end("Hello Node.js");
9  });
10
11 /* Bind server to a port */
12 server.listen(8080);
```

Listing 19: Node.js - HTTP Module (`server.js`): using es6 features

Behind the scenes, `http.createServer()` is instantiating an object of `http.Server` and registers a callback function to the `request` event.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

http.Server

- Properties
  - listening
  - maxHeadersCount
  - timeout
- Methods
  - close()
  - listen()
  - setTimeout()
- Events
  - close
  - connect
  - connection
  - listening
  - request
  - upgrade

Objects of the class `http.Server` have different properties, methods and emit different events, to which callbacks can be registered.

You can read the detailed documentation on the HTTP module at the Node.js page for a full list of properties, methods and events.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

Method `listen()` can be used in various ways. Currently, our script does not indicate any actions happening when started from the terminal.

In addition to the `port`, we may pass a `host` name and a callback function in addition. The callback function is invoked after the server binds successfully to the specified interface and port number.

```
8  /* Bind server to a hostname and port */
9  const port = 8080;
10 const host = "127.0.0.1";
11 server.listen(port, host, () => {
12   console.log(`Server running at http://${host}:${port}/`);
13 });
```

Listing 20: Node.js - HTTP Module (`server.js`): listen callback

Using `127.0.0.1` binds the server only to the loopback interface. You can use the IP address of another inteface or use `0.0.0.0` to bind to the IP addresses of all interfaces available on your machine.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

The two parameters `request` and `response`, passed to the callback, are of objects of different classes, providing a different set of properties and methods.

```
3  /* Create a http server */
4  const server = http.createServer((request, response) => {
5    console.log({
6      request, // http.IncomingMessage
7      response // http.ServerResponse
8    });
9    response.end("Hello Node.js");
```

Listing 21: Callback parameter: `request` and `response`

You can inspect the properties of those two parameters by logging them to the console or by using the developer tools of the browser.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

http.ServerResponse

- Properties
  - statusCode
  - statusMessage
- Methods
  - setHeader()
  - write()
  - end()
- Events
  - close
  - finish

The setHeader() method is used to specify the header of the response, whereas write() is used to send the body of the response in chunks.

The end(), finally, completes the response, accepting a final chunk for the body.

Further calls of write() and end() are not allowed for the current event.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Module

http.IncomingMessage

- Properties
  - `headers`
  - `httpVersion`
  - `method`
  - `statusCode`
  - `statusMessage`
- Methods
  - `destroy()`
  - `setTimeout()`
- Events
  - `data`
  - `aborted`
  - `end`
  - `close`

The properties `statusCode`, `url`, and `method` contain the associated data of the request, respectively.

Property `headers` is an object which contains the header of the HTTP message as key-value pairs

Objects of this implement the *Readable Stream* interface `stream.Readable`. As such, they emit events like `data` and `end`.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

To respond to a client request with a HTML page, we can call `writeHead()` and `write()` to write out data immediately, in small chunks, allowing the browser to start processing the first parts of the page without waiting to receive the complete message.

```
3  /* Create a http server */
4  const server = http.createServer((request, response) => {
5    response.writeHead(200, { "Content-Type": "text/html" });
6    response.write("<html>");
7    response.write("<head><title>My First Page</title></head>");
8    response.write("<body>");
9    response.write("<h1>Hello from my Node.js Server!</h1>");
10   response.write("</body>");
11   response.write("</html>");
12   response.end();
13 });
```

Listing 22: Node.js - HTTP Response

Finally, we have to call `end()` to complete the transmission.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

If you open `http://localhost:8080` in your browser, you should receive the expected response.

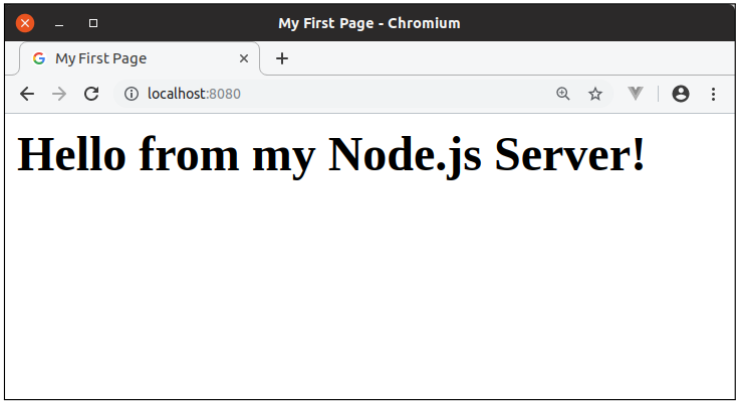

Figure 16: Receiving a HTML page from the server

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

Alternatively, we could have created a (template) strings and use `setHeader()` to send the page as a whole. The drawback of this approach is, however, that the client won't receive anything to process (and render) in advance.

```
3  /* Create a http server */
4  const server = http.createServer((request, response) => {
5    const page = `<html>
6        <head><title>My First Page</title></head>
7        <body><h1>Hello from my Node.js Server!</h1></body>
8      </html>`;
9    response.setHeader("Content-Type", "text/html");
10   response.end(page);
11 });
```

Listing 23: Node.js - HTTP Response

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

You can inspect the details of the request and associated response in the developer tools of your browser.

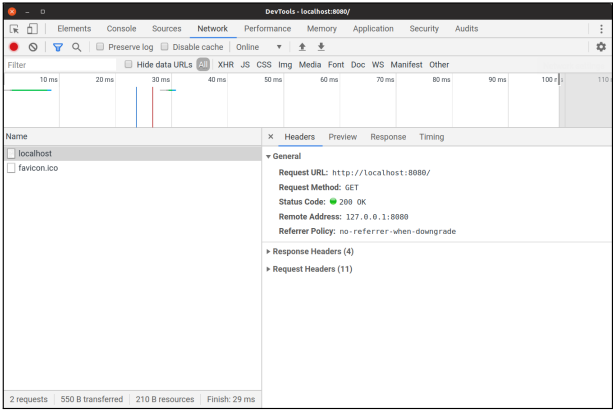

Figure 17: General request/response information shown in the developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

The first entry in the list on the left side is the actual request that was the result of entering an URL in the browser address bar.

Your browser than tries to fetch a file called `favicon.ico`, automatically. The file would be used as an icon for your list of *Favorites* and on other occasions.

The previous figure (Figure 17) shows the *Network* tab of the developer tools. It displays general HTTP header information about the selected entry. You can read more about the available features online (Chrome, Firefox).

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

Your browser sent various information in the HTTP header of its request, which could be used by your application.

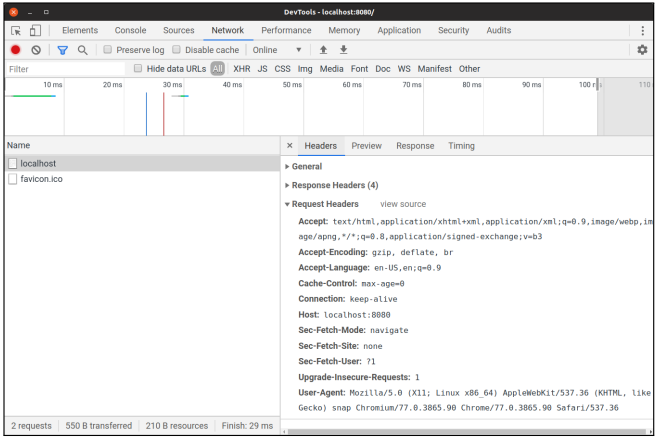

Figure 18: HTTP request header

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

Your application can use the HTTP header of the response to instruct the browser or pass information to the client application.

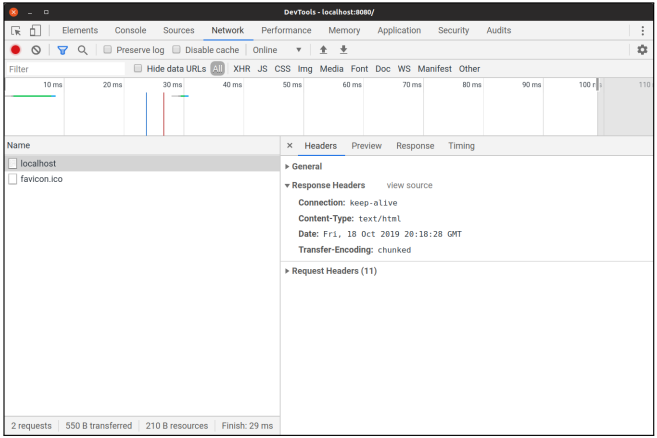

Figure 19: HTTP response header

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

Besides from information on request and response header, you can take see the raw data and a preview of the response body.

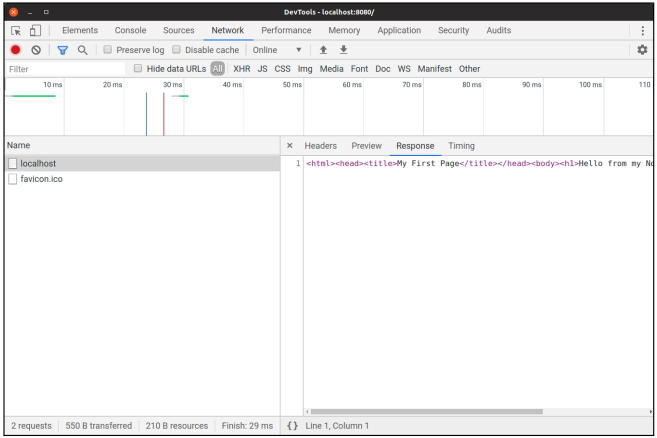

Figure 20: HTTP response body

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Response

The developer tools usually allow to inspect the timing in more detail. For further details, please refer to the documentation.

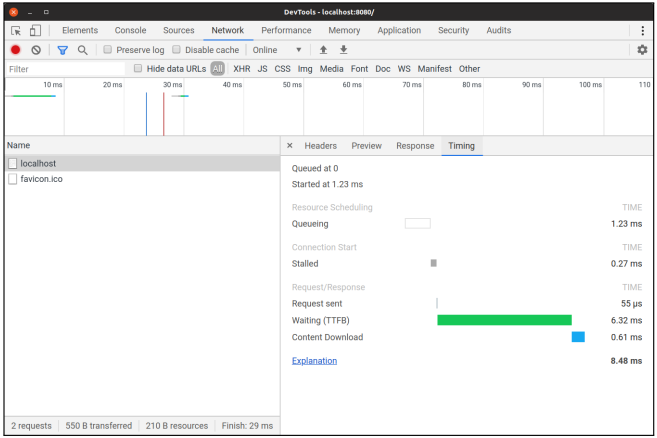

Figure 21: HTTP response timing

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Routing

So far, our server responds always with the page, no mather what resource is requested by the client.

For instance, when you enter `http://localhost:8080/other.html`, requesting file `other.html`, the server will respond with the same welcome page that you get with `http://localhost:8080/`.

Figure 22: Requesting a different HTML page from the server

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Routing

We can use the first parameter (`request`) of the callback function to *route* the client to different pages.

```
1  const http = require("http");
2
3  const welcomePage = `<html>
4    <head><title>My First Page</title></head>
5    <body><h1>Hello from my Node.js Server!</h1></body>
6  </html>`;
7  const defaultPage = `<html>
8    <head><title>My First Page</title></head>
9    <body>
10     <h1>This Page is not available!</h1>
11     <h2>Take me <a href="/index.html">home</a>.</h2>
12   </body>
13 </html>`;
14
15 /* Create a http server */
16 const server = http.createServer((request, response) => {
17   response.setHeader("Content-Type", "text/html");
18   if (request.url === "/index.html") {
19     return response.end(welcomePage);
20   }
21   response.statusCode = 404;
22   response.end(defaultPage);
23 });
```

Listing 24: Node.js - Routing

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Routing

All clients that target the index page, that is `index.html`, will get a welcome page shown.

Clients requesting any other resources will get a default page as a response, including a hyperlink to the index page.

Note, that we have to `return` after calling `end(welcomePage)`. Otherwise, the second call to `end(defaultPage)` will result in an error thrown, because no further calls of it are allowed for this request.

If not otherwise specified, the HTTP status code send to the client, is implicitly set to 200. We can use status code 404 to tell clients, that the requested resources does not exist (*Not Found*), when responding with the default page.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Routing

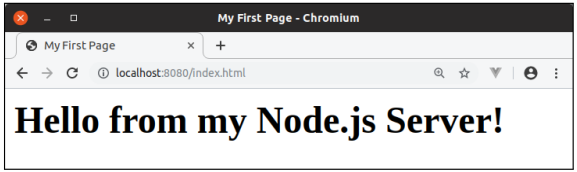

Figure 23: Requesting `index.html` from the server

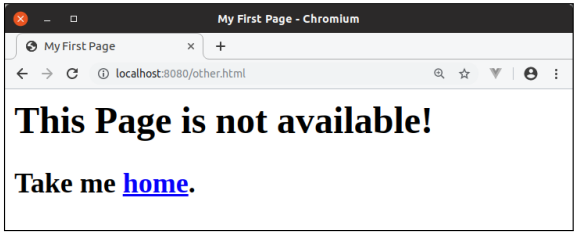

Figure 24: Requesting other resources from the server

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Routing

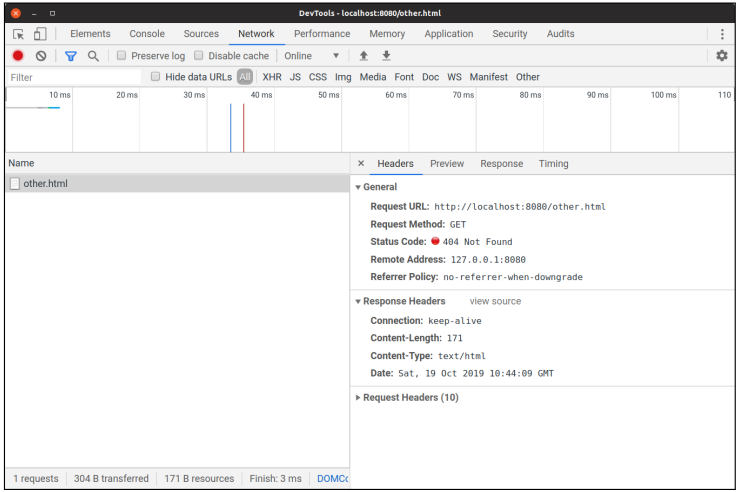

Figure 25: Requesting other resources: developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Redirecting

If the client does not specify any resource, targeting only the domain name, that is `http://localhost:8080` it will currently receive the default page.

Usually, server are set up to respond with the HTTP status 301, telling the browser, that the requested resource is at/moved to a different location. Search engines and browsers may use this information for future request and ask for `index.html` when the user enters the domain only.

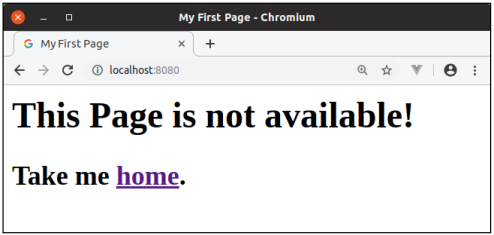

Figure 26: Entering the domain name without specifying a resource

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Redirecting

We can test for this case and use `writeHead()` reply with a status code 301 and the header field *Location*, telling the client to request the specified resource instead.
A call of `writeHead()` will immediately send a response header containing the provided status code and only the header fields provided.

```
15 /* Create a http server */
16 const server = http.createServer((request, response) => {
17   if (request.url === "/") {
18     response.writeHead(301, { Location: "/index.html" });
19     return response.end();
20   }
21   if (request.url === "/index.html") {
22     response.setHeader("Content-Type", "text/html");
23     return response.end(welcomePage);
24   }
25   response.setHeader("Content-Type", "text/html");
26   response.end(defaultPage);
27 });
```

Listing 25: Node.js - Redirecting

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Redirecting

When we now use the domain name only, that is `localhost:8080/`, we will immediately be directed to page `index.html`.

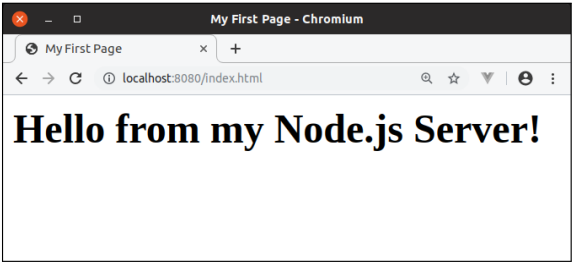

Figure 27: Requesting no resource

The browser will try to fetch both resource, which we can verify in the developer tools (Figure 28).

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Redirecting

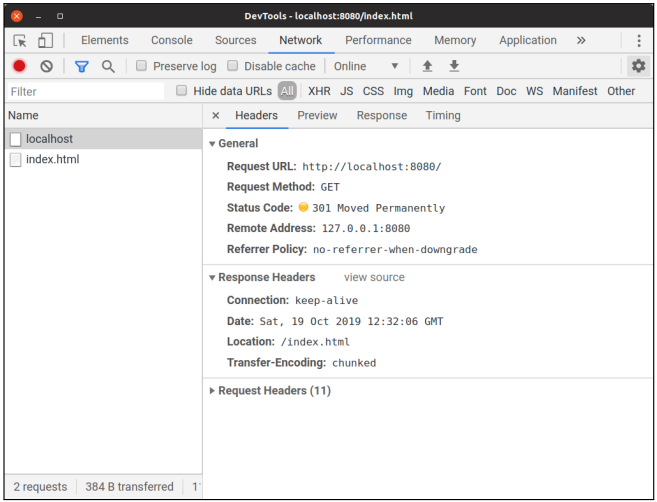

Figure 28: Requesting no resource: developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

We will extend our example now with an additional page, which allow the user to fill out a form and send the data to the server, for processing.

```
14 const todoPage = `<html>
15   <head><title>My First Page</title></head>
16   <body>
17     <form action="/todo/add" method="POST">
18       <div><input type="text" name="todo" autofocus/></div>
19       <div><button type="submit">Create</button></div>
20     </form>
21   </body>
22 </html>`;
```

Listing 26: Adding a new page: `todo.html`

When the client requests page `todo.html` it will now get the newly created page (Listing 26).

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

```
24 /* Create a http server */
25 const server = http.createServer((request, response) => {
26   const { url } = request;
27   if (url === "/") {
28     response.writeHead(301, { Location: "/index.html" });
29     return response.end();
30   } else {
31     response.setHeader("Content-Type", "text/html");
32     if (url === "/index.html") {
33       return response.end(welcomePage);
34     } else if (url === "/todo.html") {
35       return response.end(todoPage);
36     }
37   }
38   response.statusCode = 404;
39   response.end(defaultPage);
40 });
```

Listing 26: Adding a new page: `todo.html` (continued)

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

The new page `todo.html` consits only of a form, where users can enter some text, and when done, press the submit button.

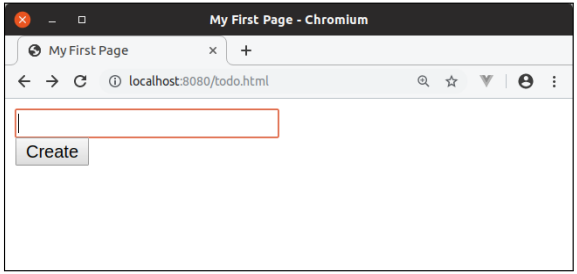

Figure 29: Requesting the new page

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Pressing the submit button will result in a post request sent to the server, targeting route `/todo/add` and including the value of the input field in the body.

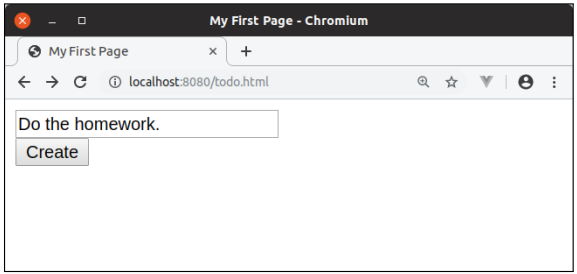

Figure 30: Filling out the form and sending a POST request

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

The server, however, responds with the default page and status code 404.

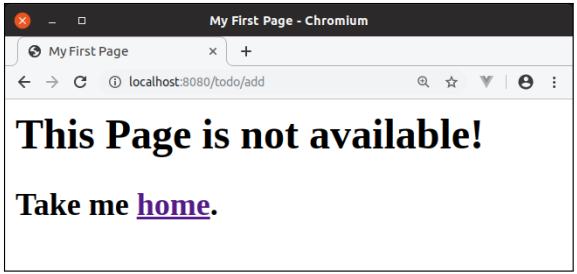

Figure 31: Response from server on the previous POST request

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

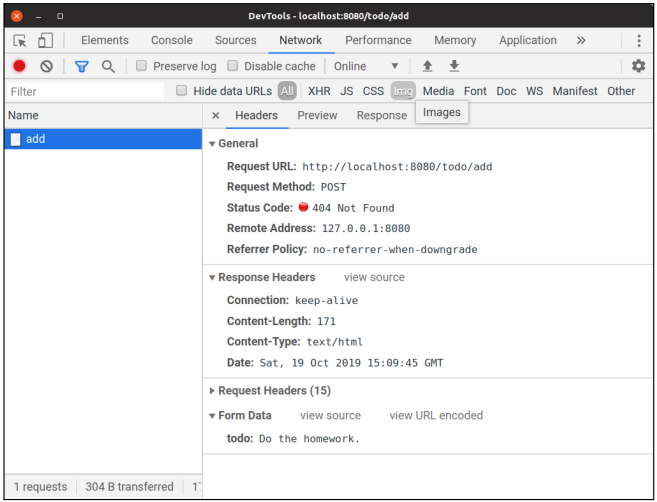

Figure 32: Response from server on the previous POST request: developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

To change this behavior, we have to add another case for the route the request was posted to.

```
32      if (url === "/index.html") {
33        return response.end(welcomePage);
34      } else if (url === "/todo.html") {
35        return response.end(todoPage);
36      } else if (url === "/todo/add") {
37        console.log(request.method);
38        response.writeHead(302, { Location: "/" });
39        return response.end();
40      }
```

Listing 27: Adding a new route: `/todo/add`

This way, we log the used HTTP method to the console log and redirect the browser to the root of the domain, this time using status code 302.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

The browser now gets redirected twice. First to the root of the domain, by posting to route `/note/add`, indicating that the resource was moved temporarily. And the second time to page `index.html`, by the root of the domain, this time indicating that the requested resource was moved permanantly.

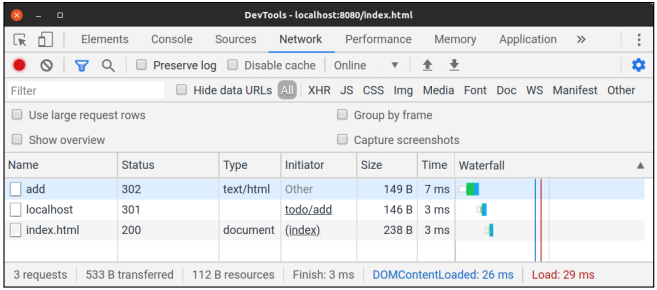

Figure 33: Response from server on the previous POST request: developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Typically, you don't want unnecessary redirects. The current code, however, is only for demonstration.

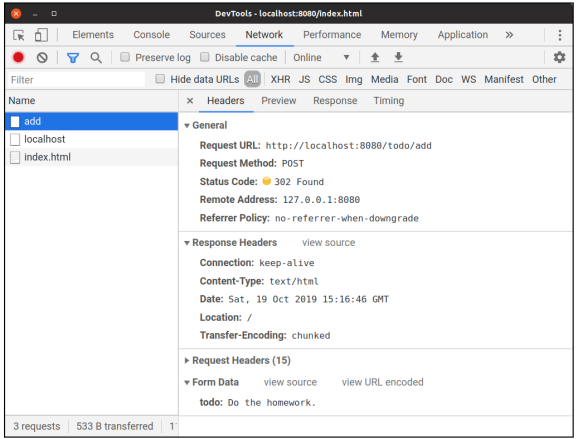

Figure 34: Response from server on the previous POST request: developer tools

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

After you restart Node.js to reload the updates of the script, you should see the HTTP method logged to the console, each time the browser requests the associated route.

```
~ $ node server.js
Server running at http://127.0.0.1:8080/
POST
POST
GET
```

Listing 28: Server console log for client requests to `/todo/add`

Can you think of a way to get method GET logged to the console, without the changing the current code?

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Processing the data send by the client is rather involved.

```
26   const { url, method } = request;
27   if (url === "/") {
28     response.writeHead(301, { Location: "/index.html" });
29     return response.end();
30   } else {
31     response.setHeader("Content-Type", "text/html");
32     if (url === "/index.html") {
33       return response.end(welcomePage);
34     } else if (url === "/todo.html") {
35       return response.end(todoPage);
36     } else if (url === "/todo/add" && method === "POST") {
37       const data = [];
38       request.on("data", chunk => data.push(chunk));
39       request.on("end", () => console.log(data));
40       response.writeHead(302, { Location: "/" });
41       return response.end();
42     }
43   }
```

Listing 29: Processing the HTTP body of the client request

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Callback paramter `request` does not have any additional property like `url` or `method`, providing the data sent by the client.

We can, however, register two callbacks to the `request` parameter. Event `data` is called for each chunk of data received by the client.

```
38      request.on("data", chunk => data.push(chunk));
```

This callback function gets each chunk of data passed as the first argument.

Event `end`, on the other hand, is called at the end of each request.

```
39      request.on("end", () => console.log(data));
```

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Node.js receives the client request as a stream of incoming data. This allows server application to start processing the data as the first parts coming in, without the need to wait for the complete response.

Although the extra effort might seem odd, since we are only receiving the data of one input field, imagine the simple case of the upload of a larger file. We can immediately start writing the first parts of the upload.

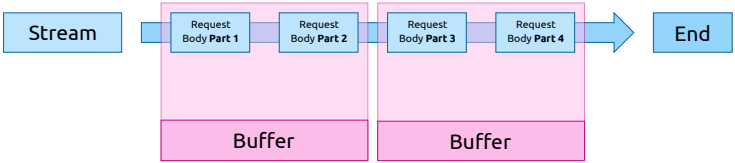

Figure 35: `Stream` & `Buffer` API

The chunks of data passed to the callback for `data` events, are of type `Buffer`, which is a Node.js mechanism to deal with binary data of those streams.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

After you restart Node.js and load the updated script, it will log the received chunks to the console.

```
~ $ node server.js
Server running at http://127.0.0.1:8080/
[ <Buffer 74 6f 64 6f 3d 44 6f 2b 74 68 65 2b 68 6f 6d 65 77 6f 72 6b 2e> ]
[ <Buffer 74 6f 64 6f 3d 44 6f 2b 74 68 65 2b 68 6f 6d 65 77 6f 72 6b 2e> ]
[ <Buffer 74 6f 64 6f 3d 47 6f 2b 74 6f 2b 73 6c 65 65 70 25 32 31> ]
```

Listing 30: Logging the received HTTP body to the console

Each array logged to the console is the result of a single request. Since we are only receiving small amounts of data, we only get one `Buffer` object, hence one object in the array.

We may transform the array of `Buffer` objects to a single `Buffer` object, instead of processing the each chunk, in the **end** callback.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Finally, we need a way to convert to `Buffer` object to a string.

```
38      request.on("data", chunk => data.push(chunk));
39      request.on("end", () => {
40        const buffer = Buffer.from(...data);
41        console.log(buffer.toString());
42      });
```

Listing 31: Converting the `Buffer` array to a string

Alternatively, we could have achieved the same result with the following approach.

```
38      request.on("data", chunk => data.push(chunk.toString()));
39      request.on("end", () => console.log(data.join()));
```

Listing 32: Converting single `Buffer` objects to strings

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

When you restart the server again and enter some notes to be created, you should see the output as string key-value pairs.

```
~ $ node server.js
Server running at http://127.0.0.1:8080/
todo=Do+the+homework.
todo=Do+the+homework.
todo=Go+to+sleep%21
```

Listing 33: Logging the processed HTTP body to the console

The result we receive is percent-encoded and needs to be decoded (including additional replacements) for correct presentation.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

To complete the request processing example, we can write the client response to a file.

```
37      } else if (url === "/todo/add" && method === "POST") {
38        let data = "";
39        request.on("data", chunk => (data += chunk.toString()));
40        request.on("end", () => {
41          addTodo(data);
42          response.writeHead(302, { Location: "/" });
43          return response.end();
44        });
45        return;
46      }
```

Listing 34: Processing request and writing data to a file

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

We convert the received chunks of type `Buffer` to strings, which are appended to `data`.

```
41          addTodo(data);
42          response.writeHead(302, { Location: "/" });
43          return response.end();
```

When the transmission of the request finishes, `addNote(data)` is called, which processes the received body, and a response, redirecting the client, is sent.

Note that we need to return inside this `if` branch, otherwise the server will send the default page to the client.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Inside of `addTodo()` we process the received user data, to the *note* sent by the client.

```
52 /* Add a new todo note */
53 function addTodo(data) {
54   const filename = "./todo.txt";
55   const note = processUserData(data);
56   fs.exists(filename, exists => {
57     if (exists) updateNotes(filename, note);
58     else createNotes(filename, note);
59   });
60 }
```

Listing 35: Processing request and writing data to a file

We then test update an existing file or create an one, and depending on whether it exists or not.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

To parse the note from the data, we first pass the received string to `decodeURIComponent()` to decode the percent-encoded data. Node.js provides this function, like other function known from the browser API.

```
61 function processUserData(data) {
62   return decodeURIComponent(data)
63     .replace(/(.*?)=(.*)/g, (m, key, value) => value)
64     .replace(/\+/g, " ");
65 }
```

Listing 36: Parsing the note from the request body

Afterwards, the decoded string is converted using the string function `replace()` and regular expressions, using a special syntax.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

The first call, seperates the key and value on the first occurrence of an equal sign, using capture groups.

```
63      .replace(/(.*?)=(.*)/g, (m, key, value) => value)
```

The second call, simply replaces the plus sign, used instead of a space character.

```
64      .replace(/\+/g, " ");
```

Note, currently, users can't enter any plus sign inside their notes. This was not implemented for the sake of clarity. To correct this, simply replace the plus sign with a space character, before passing the string to `decodeURIComponent()`. Plus signs entered by the user will be percent-encoded and thus not targeted by `replace()`.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - HTTP Request

Function `updateNotes()` reads the file, as an `utf-8` string, combines the content of the file with the new note, and passes the result to `createNotes()`.

```
66 function updateNotes(filename, note) {
67   fs.readFile(filename, "utf-8", (error, content) => {
68     if (!error) createNotes(filename, [content, note].join("\n"));
69     else console.error(`Could not read file ${filename}.`);
70   });
71 }
72 function createNotes(filename, note) {
73   console.log(`Creating note ${filename}:`, "\n", note);
74   fs.writeFile(filename, note, error => {
75     if (error) {
76       console.error(`Could not create file ${filename}.`);
77       console.error(error);
78     }
79   });
80 }
```

Listing 37: Writing the note to a file

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Modules

You can create two new files, `routes.js` and `pages.js` and place some parts of the server application inside of them, to organize the source code.

```
1  const http = require("http");
2  const { router } = require("./routes.js");
3
4  /* Create a http server */
5  const server = http.createServer(router);
6
7  /* Bind server to a hostname and port */
8  const port = 8080;
9  const host = "127.0.0.1";
10 server.listen(port, host, () => {
11   console.log(`Server running at http://${host}:${port}/`);
12 });
```

Listing 38: Organizing the source code with modules

The main entry file for the server application, setting up the server, is now reduced to a few single lines.

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Modules

The module provided by `pages.js` uses the mechanism provided by `module.exports` to export an object containing all pages.

```
1  const welcomePage = `<html>
2    <head><title>My First Page</title></head>
3    <body><h1>Hello from my Node.js Server!</h1></body>
4  </html>`;
5  const defaultPage = `<html>
6    <head><title>My First Page</title></head>
7    <body>
8      <h1>This Page is not available!</h1>
9      <h2>Take me <a href="/index.html">home</a>.</h2>
10   </body>
11 </html>`;
12 const todoPage = `<html>
13   <head><title>My First Page</title></head>
14   <body>
15     <form action="/todo/add" method="POST">
16       <div><input type="text" name="todo" autofocus/></div>
17       <div><button type="submit">Create</button></div>
18     </form>
19   </body>
20 </html>`;
21
22 module.exports = { welcomePage, defaultPage, todoPage };
```

Listing 39: Node.js module containing the pages

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Node.js - Modules

Most of our application consists of the routing algorithm provided by `routes.js`. Inside of `routes.js` we have to import the pages provided by `pages.js`.

```
1 const fs = require("fs");
2 const { welcomePage, defaultPage, todoPage } = require("./pages.js");
```

Listing 40: Node.js module imports used by the router

The rest of the code is basically the same. The only difference is the change from arrow function expressions to a function definition (Listing 41) and the export of this function (Listing 42).

```
4 function router(request, response) {
```

Listing 41: Router function defintion

```
60 module.exports = { router };
```

Listing 42: Node.js module exports of router function

Alija Sabic
Web Development

Node.js
History
Installation
Applications
HTTP Module
Response
Routing
Redirecting
Request
Files
Modules

Acronyms I

DNS	Domain Name System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LTS	Long Term Stable
MSI	Microsoft Installer
`npm`	Node Package Manager
`nvm`	Node Version Manager
OS	Operating System
PhD	Doctor of Philosophy (philosphiae doctor)

Alija Sabic
Web Development

Acronyms
References

Acronyms II

REPL	Read Eval Print Loop
TCP	Transmission Control Protocol
TSC	Technical Steering Committee
URL	Uniform Resource Locator

Alija Sabic
Web Development

Acronyms
References

References I

[1] Ryan Dahl, "10 Things I Regret About Node.js," https://www.youtube.com/watch?v=M3BM9TB-8yA, 2018.

[2] S. Springer, *Node.js: Das umfassende Handbuch*, 3rd ed. Bonn: Rheinwerk Verlag, 2018. [Online]. Available: http://d-nb.info/1168478928

[3] Wikipedia contributors, "Ryan Dahl," https://en.wikipedia.org/w/index.php?title=Ryan_Dahl, 10 2019.

[4] Node.js Foundation, "Node.js - JavaScript Runtime," https://nodejs.org/, 2019.

References II

[5] Ryan Dahl, “Node.js,” https://www.youtube.com/watch?v=EeYvFl7li9E, 2009.

Alija Sabic
Web Development

Acronyms
References