NODEJS INTRO

AGENDA

- Node.js overview
- Node.js modules
- NPM
- Package.json file
- NodeJS installation
- First Node.js application
- Request and response methods

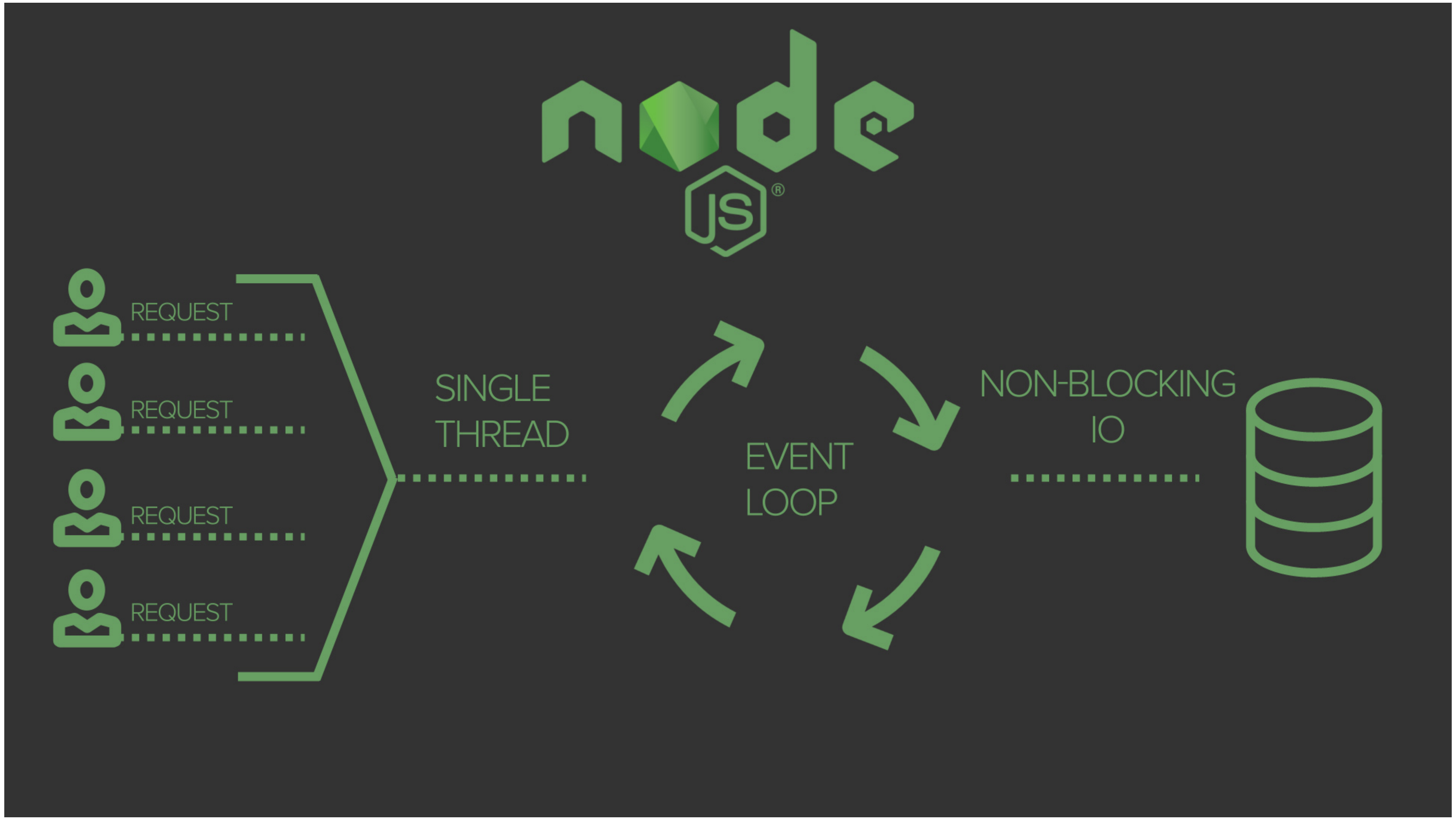
WHAT NODE.JS IS

- Created in 2009, Open-sourced for now.
- JavaScript runtime. Not a language or a framework
- Runs on v8 JavaScript engine, same a Google Chrome
- Written on C++ & JavaScript
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server
- Node.js = Runtime Environment + JavaScript Library

FEATURES OF NODE.JS

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, **non-blocking**. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

NODE.JS ARCHITECTURE



Node.js is ideal for I/O-intensive apps

softserve

http://latentflip.com/loupe

ADVANTAGES

- Fast and event-based
- Scalable
- Rich ecosystem

DISADVANTAGES

- Not suited for CPU-intensive tasks
- Asynchronous model is difficult to learn and understand
- API is not super-stable

softserve

USE NODE.JS FOR:

- **Front-end** (Webpack, front-end tools, .etc)
- **Back-end** (API, microservices, REST/GraphQL/Sockets…)
- **Desktop apps** (Electron: Slack, Atom, VS Code, WhatsApp)
- **Bots**
- **IoT** (Cylon.js/JohnnyFive)
- **CLI**

DON'T USE NODE.JS FOR:

- **CPU-heavy jobs**
- **Image processing**
- **BigData processing / Math**

softserve

WHO USE NODE.JS

NETFLIX

Trello

LinkedIn

UBER

Medium

PayPal

ebay™

NASA

NODE.JS MODULES

- Node.js uses a modular system. That is, all built-in functionality is divided into separate packages or modules.
- A module is a block of code that can be reused in other modules.
- Consider modules to be the same as JavaScript libraries.
- Node.js has a set of built-in modules which you can use without any further installation.
- To include a module, use the ***require()*** function with the name of the module:

```
const url = require('url');
```

NODE.JS POPULAR MODULES

- **Express** is a popular, fast Node.js framework for web and mobile application development.
- **Socket.io** - framework for building realtime applications
- **Mongo/Mongoose** – wrappers to interact with MongoDB.
- **Pug/Jade** – template engine inspired by HAML
- **Keystone.JS** - designed for building database-driven websites, applications and APIs
- **Passport** is a unique authentication module for Node.js devs
- **Nodemon** is a utility that will monitor for any changes in your source and automatically restart your server.

softserve

NPM

- NPM is a Node.js Package Manager
- NPM used to install node packages/modules
- The NPM program is installed on your computer when you install Node.js (to check the NPM version use “***npm -v***”)
- NPM creates a folder named "***node_modules***", where the package will be placed.

```
npm install <module_name>        // how to install any Node.js module
npm install express              // install a package locally
npm install -g express           // install a package globally
```

softserve

PACKAGE.JSON FILE

- all dependencies are listed in a “***package.json***” file
- package.json is present in the root directory of any Node application/module

npm init // create a package.json file

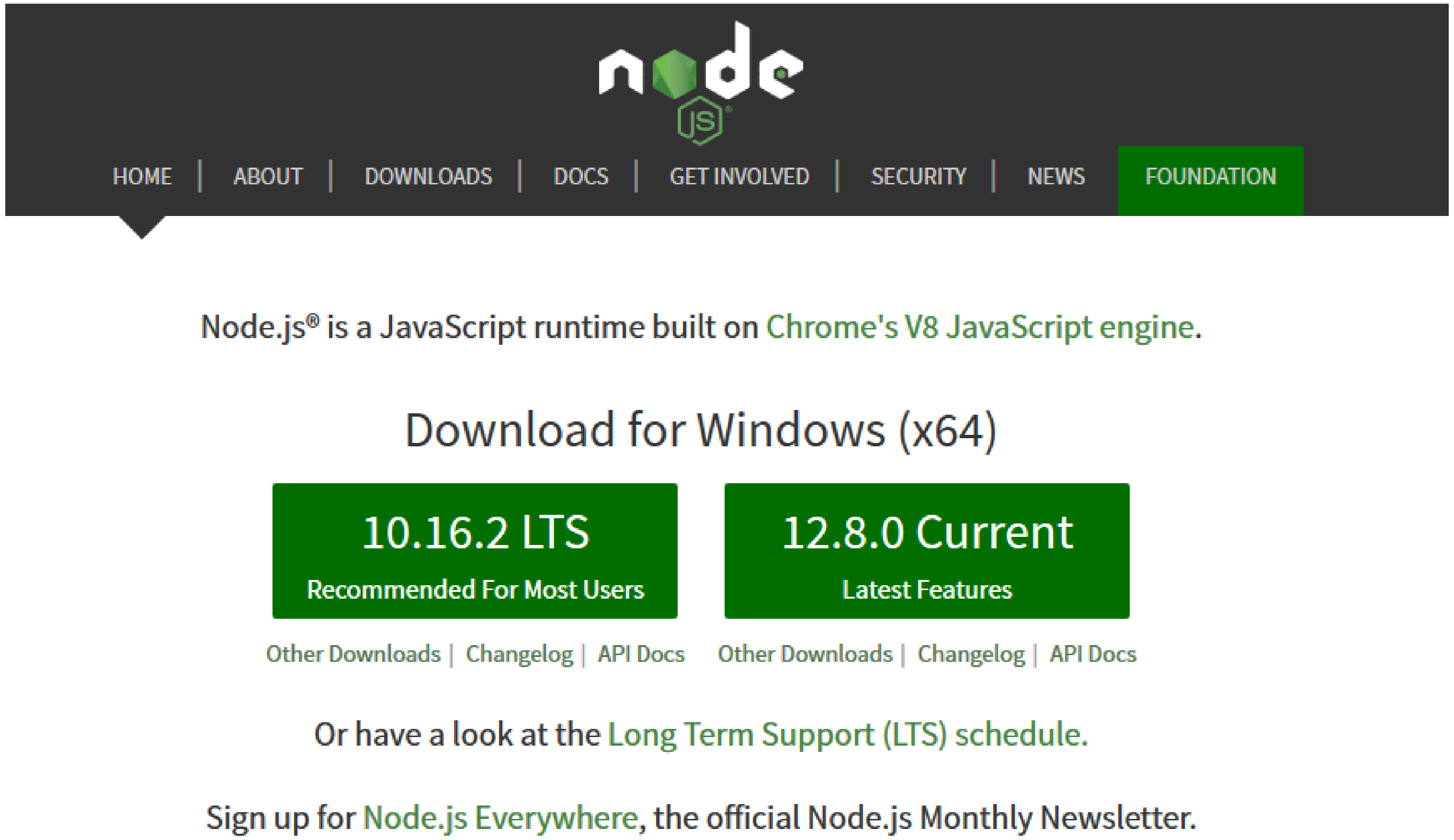
```
{
   "name": "node-js-sample",
   "version": "0.2.0",
   "description": "A sample Node.js app using Express 4",
   "main": "index.js",
   "author": "Mark Pundsack"
   "dependencies": {
       "express": "^ 4.13.3“,
       "mongojs": "^ 2.4.0"
   }
}
```

softserve

NODEJS INSTALLATION

1. Download installation package from https://nodejs.org/ . For Windows, this is a file with the *msi* extension.

2. If you have a different operating system, select *Other Downloads* and download the required installation package.


node js
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

10.16.2 LTS
Recommended For Most Users

12.8.0 Current
Latest Features

Other Downloads | Changelog | API Docs   Other Downloads | Changelog | API Docs

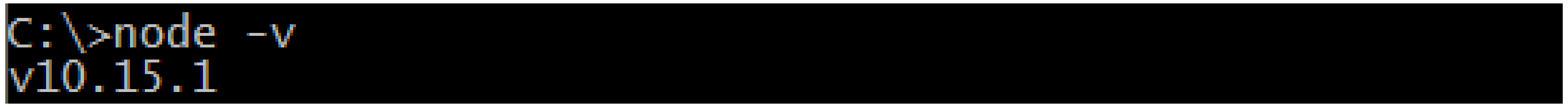
Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

softserve

VERIFY INSTALLATION: CHECK VERSION & EXECUTING A FILE

•1. After a successful installation, you can enter the ***node -v*** command on the command line / terminal and the current version of node.js will be displayed:

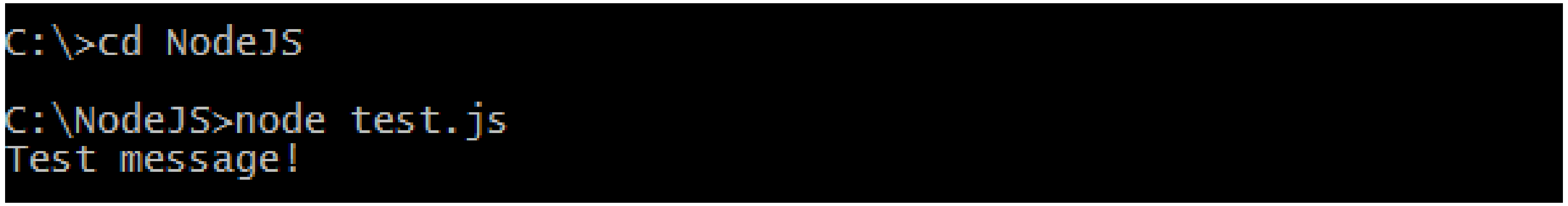


```
C:\>node -v
v10.15.1
```

2. Create a js file named ***test.js*** on your PC in directory, for example, ***NodeJS*** having the following code:

```
console.log("Test message!");
```

At the command prompt, use the ***cd*** command to navigate to the ***NodeJS*** directory, and then run the command ***node test.js***, which will execute the code from the ***test.js*** file:



```
C:\>cd NodeJS

C:\NodeJS>node test.js
Test message!
```

softserve

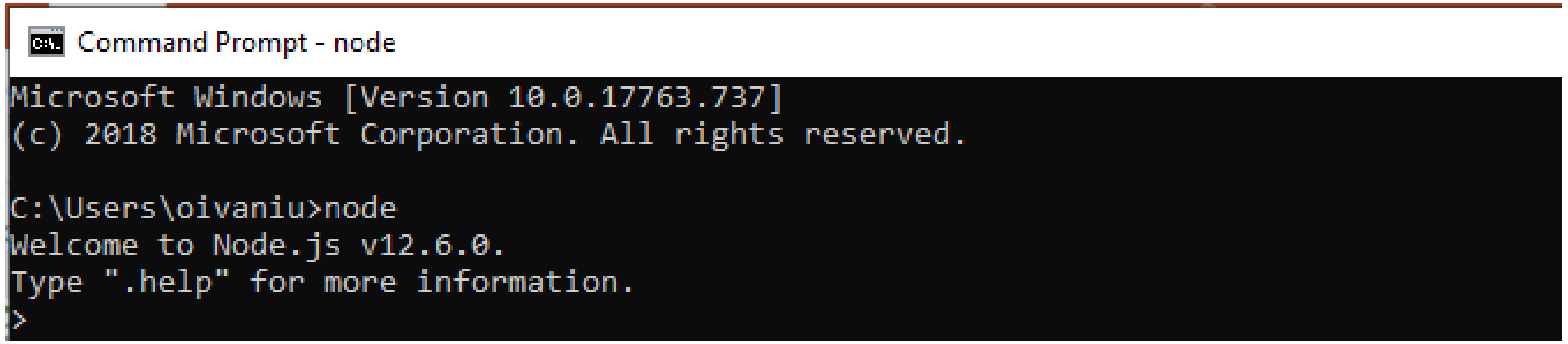
NODE.JS REPL Terminal

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode.

Node.js comes bundled with a REPL environment. It performs the following tasks:

- **Read** – Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- **Eval** – Takes and evaluates the data structure.
- **Print** – Prints the result.
- **Loop** – Loops the above command until the user presses **ctrl-c** twice.

REPL can be started by simply running ***node*** on shell/console without any arguments:


Command Prompt - node

```
Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\oivaniu>node
Welcome to Node.js v12.6.0.
Type ".help" for more information.
>
```

softserve

WORK WITH FILE SYSTEM

To work with the file system, we need to use the *fs* module.

Read files:

1) **fs.readFile**(path [, options], callback)
2) **fs.readFileSync**(path [, options])

- *path* - filename
- *options* – encoding type
- *callback* - function which pass two arguments (err, data), where data is the contents of the file.

Write files:

fs.writeFile(file, data [, options], callback)

softserve

FIRST NODE.JS APPLICATION

A Node.js application consists of the following three important components:

- **Import required modules** − We use the ***require*** directive to load Node.js modules.

- **Create server** − A server which will listen to client's requests similar to Apache HTTP Server.

- **Read request and return response** − The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

softserve

1) IMPORT REQUIRED MODULE

- In the browser, when we want to add a JS file to the page, we use the *<script>* tag, and in NodeJS *require*. In essence, a module is a file that is connected using *require*.
- So, we use the ***require*** directive to load the http module and store the returned HTTP instance into an http variable as follows:

```
const http = require("http");
```

2) CREATE SERVER

We use the created http instance and call **http.createServer()** method to create a server instance and then we bind it at port 8000 using the **listen()** method associated with the server instance. Pass it a function with parameters request and response.

```
http.createServer( function(request, response) {
  // Send the HTTP header with HTTP Status: 200=OK, Content Type: text/plain
  response.writeHead(200, {'Content-Type': 'text/html'});

  // Send the response body as "Testing NodeJS server"
  response.end('Testing NodeJS server\n');
}).listen(8000);

// Console will print the message
console.log('Server running at http://127.0.0.1:8000/');
```

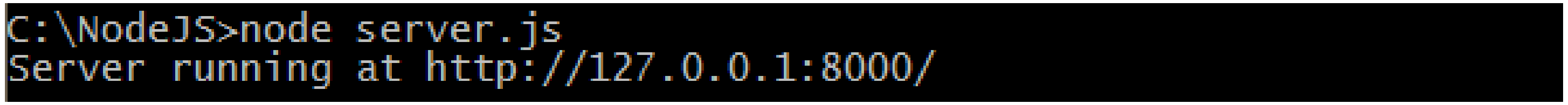
softserve

3) TESTING REQUEST & RESPONSE

Import the required module and create a server in the *server.js* file and start our HTTP server as shown below:

```
const http = require("http");
http.createServer(function (request, response) {
   response.writeHead(200, {'Content-Type': 'text/plain'});
   response.end('Testing NodeJS server\n');
}).listen(8000);
console.log('Server running at http://127.0.0.1:8000/');
```

Now execute the *server.js* to start the server and verify the output:

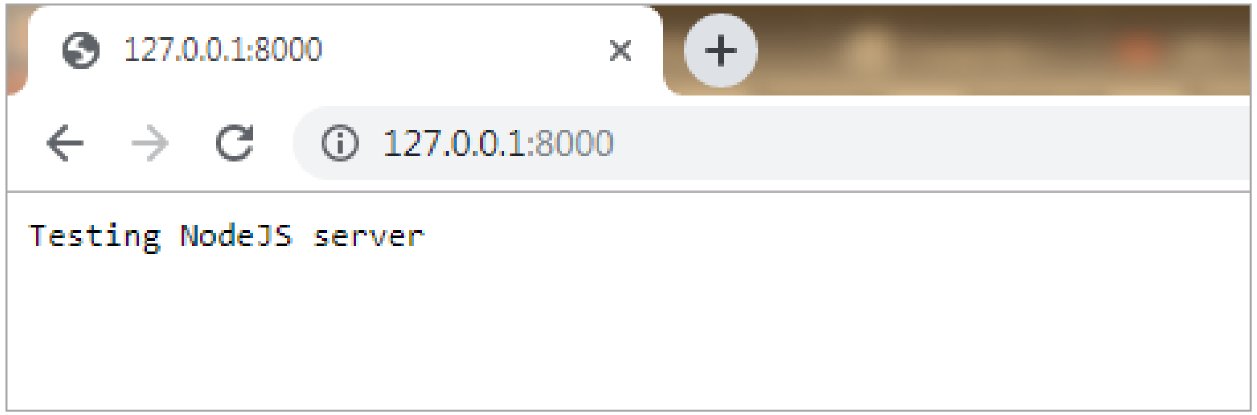


```
C:\NodeJS>node server.js
Server running at http://127.0.0.1:8000/
```

softserve

REQUEST TO THE NODE.JS SERVER

Open http://127.0.0.1:8000/ or http://localhost:8000 in any browser and observe the following result.



```
Testing NodeJS server
```

softserve

REQUEST METHODS

The **request** parameter provides information about the request and represents the ***http.IncomingMessage object***. We note some basic properties of this object:

- **headers**: returns request headers
- **method**: request type (GET, POST, DELETE, PUT)
- **url**: represents the requested address

softserve

RESPONSE METHODS

The **response** parameter controls the response and represents the ***http.ServerResponse object***. Among its functionality, the following methods can be distinguished:

- **statusCode**: sets the response status code
- **statusMessage**: sets the message sent with the status code
- **setHeader(name, value)**: adds one header to the response
- **write**: writes some content to the response stream
- **writeHead**: adds a status code and a set of headers to the response
- **end**: signals to the server that the headers and body of the response are set, as a result, the response is sent to the client. This method should be called in each request.

softserve

USEFUL LINKS

https://nodejs.org

https://www.tutorialspoint.com/nodejs/index.htm

https://www.w3schools.com/nodejs/

https://medium.com/webbdev/js-db3d35ffed7e

THANKS

softserve