



# Node.js

## Web server creation

# Web Server

- Web server can refer to
  - hardware
  - software
  - both of them working together

# Web Server

- **hardware side**

- a web server is a computer that stores **web server software** and a **website's component files** (ie. HTML docs, images, CSS & JS files)
- a web server *connects* to the Internet and supports *physical data interchange* with other devices connected to the web.

# Web Server

- **Software side**

- a web server includes several parts that control **how** web users access hosted files. At a minimum, this is an HTTP server.

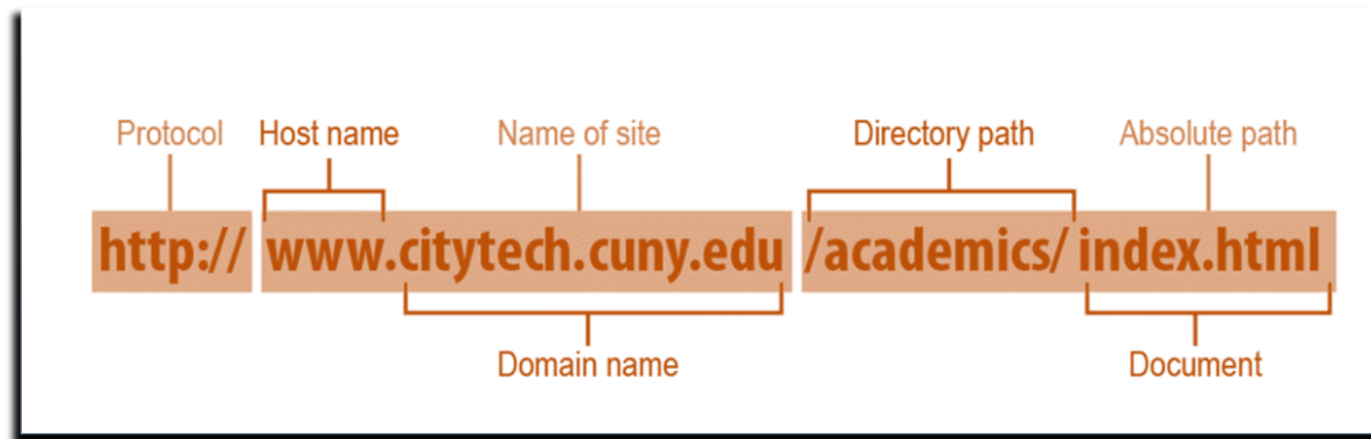
HTTP is a communication protocol (communication protocol :is a system of rules that allows two or more parties to communicate.

- **An HTTP server**

- software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages).
- a protocol that allows clients & web servers to communicate

# Web Server

- An HTTP server can be accessed through the **domain names** of the websites it stores, and it delivers the content of these hosted websites to the end user's device. ([source](#))

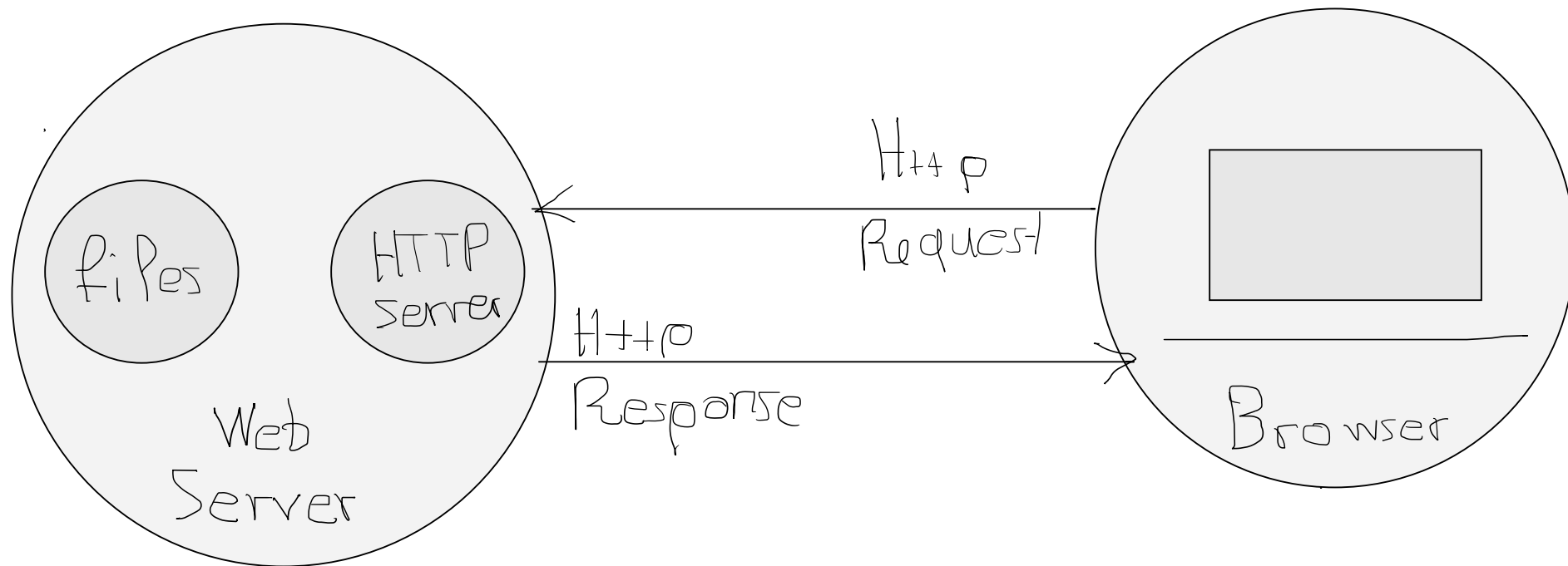


`https://stuyhsdesign.wordpress.com/web-design/domain-name/`

# Web Server

- So this is how a web server basically **works**
  - When a browser needs a file that is hosted on a web server, the browser **requests** the file via HTTP
  - When the request reaches the correct (hardware) web server, the (software) HTTP server **accepts** the request, **finds** the requested document, and sends it back to the browser (**response**), also through HTTP. (or returns 404)

# Web Server



# Web Server

- To publish a website we need-> **static** or **dynamic** web server
- **Static web server**
  - consists of a computer (hardware) with an HTTP server (software)
  - sends its hosted files **as-is** to your browser.



# Web Server

- **Dynamic web server**

- consists of a static web server plus extra software, most commonly an application server and a database.
- **application server updates hosted files** before sending content to browser via the HTTP server.

# Dynamic web sites

## Dynamic web sites

- For example, an application server might **fill** an HTML template with content from database.
- Dynamic webpages usually consist of only a few HTML templates and a database, rather than thousands of static HTML documents.
- Easier to maintain and deliver the content.

# Node.js as a Web Server

- Lets see how we can **create a web server** capable of accepting requests and sending back responses
- Node.js has a module called HTTP-> used to transfer data over the Hyper Text Transfer Protocol (HTTP)

# Node.js as a Web Server

- HTTP module can **create** an HTTP server that
  - **listens** to server ports for requests and
  - **gives** a response back to the client

# Node.js as a Web Server

Include HTTP module:

- `const http = require('http');` 1

create an HTTP server:

- `const server = http.createServer()` 2
  - Create the server with the aforementioned method that accepts a **callback** function that will fire every time a new request hits our server

# Node.js as a Web Server

## **http.createServer()** method

- turns your computer into an HTTP server.
- generates an HTTP Server object
  - HTTP Server object can **listen to ports** on your machine and execute a **requestListener method** whenever a request is received.
  - Callback function to be executed every time the server gets a request.

HTTP Server Methods and Properties: [https://www.w3schools.com/nodejs/obj\\_http\\_server.asp](https://www.w3schools.com/nodejs/obj_http_server.asp)

# Node.js as a Web Server

Listen for incoming requests at localhost port 8080

- `server.listen(8080,'127.0.0.1',callback)` 3

# Node.js as a Web Server

```
server.listen(8080,'127.0.0.1',()=>{  
  console.log('hi! We are listening to requests! ');  
});
```

3

- 8080: port number
- Localhost (like a domain name on the web) is an alias for the IP address 127.0.0.1 -> points back to our computer that is acting as a host for our website



- If we `console.log(req)` we shall see that the **req Object** is huge, with many properties and methods

```
//1st part: create a server object
const server= http.createServer((req, res) => {
  console.log(req);
}); //the server object listens on port 8080
```

- Lets see some specific options
  - Req.url
  - Req.method (type of request, get post etc)

```
//1st part: create a server object
const server= http.createServer((req, res) => {

  console.log(req.url, req.method);
  res.end();
}); //the server object listens on port 8080
```

# Node.js as a Web Server

- Res object: response
- res.write method: write to the response
- res.end method: ending the response which then sends it to the browser

# Node.js as a Web Server

```
const http = require('http');
//create a server
//createServer method will accept a callback function that will fire
//every time a new request hits our server

//1st part: create a server object
const server= http.createServer((req, res) => {

  res.write('Hello!'); //write a response to the client
  res.end('Hello World!I am ending');
  // res. end() function is used to end the response process.
}); //the server object listens on port 8080

//2nd part: listen to incoming requests from the client
//port is a subaddress in a host
//host is : 127.0.0.1 localhost
//callback function is optional
server.listen(8080,'127.0.0.1',()=>{
  console.log('hI! We are listening to requests! ');
});
```

# Node.js as a Web Server

- Save code (first\_webserver.js)
- Initiate the file: `node first_webserver.js`
- See the result at localhost: <http://localhost:8080>
  - That way the browser knows to connect to our own computer via this particular port number
- We have a real web server running on our computer!

# Node.js as a Web Server

If you check at the terminal-> you will see that the app keeps running

Remember that in previous examples-> app was executing its code and then it stopped , exited the application.

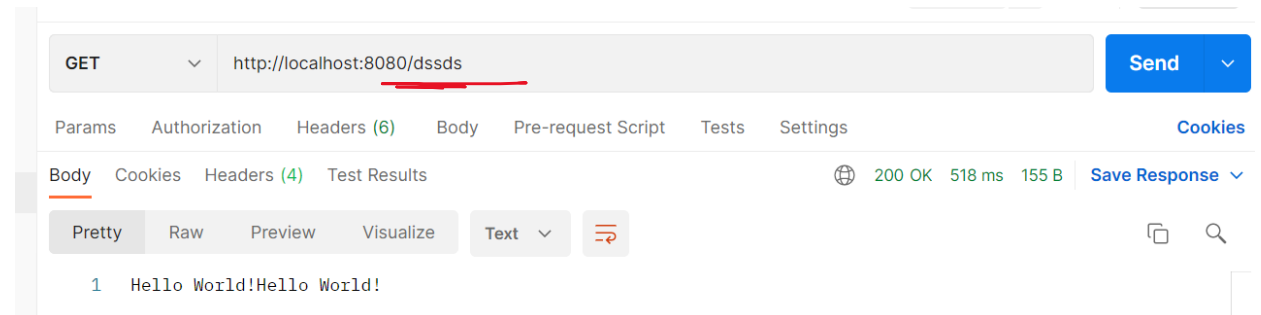
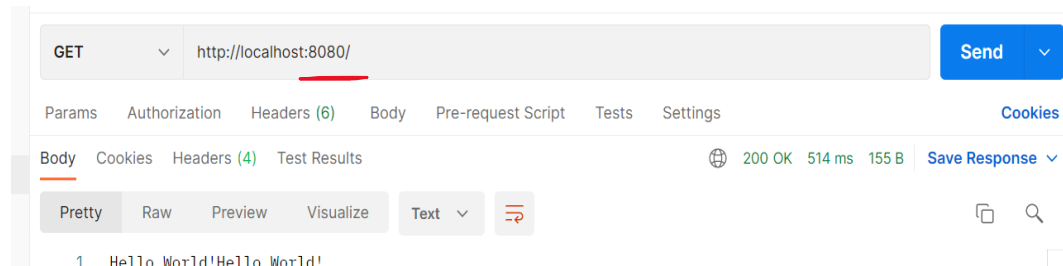
When we start a server we do not have this behavior-> the app is waiting to receive requests

# Steps...

- Step 1: We created a server, using `createServer` & passed in a **callback function**
  - This callback function
    - is executed each time a new request hits our server
    - accesses two fundamental variables: `request` & `response` variables
- Step 2: we **used** the created ***server object*** and started listening for incoming requests from the client

# Introduction

- Check that the code we wrote in our first webserver-> does not react to the URL that we're requesting.





# Introduction

## What is Routing?

- Routing defines the way in which client requests are *handled* by the application endpoints ( endpoint ie /api/users)
- **Routing actually means implementing different actions for different urls**

# Routing with node.js

- No we will see how we can implement routing without framework.
- A developer may need to build up their own server without other dependencies!
- We will know how to do so!

# URL Module

- **URL** node.js module -> splits up a **web address** into readable parts
- For very simple urls we do not actually need the url module
- We do need it for more complex ones

# Lets see an example

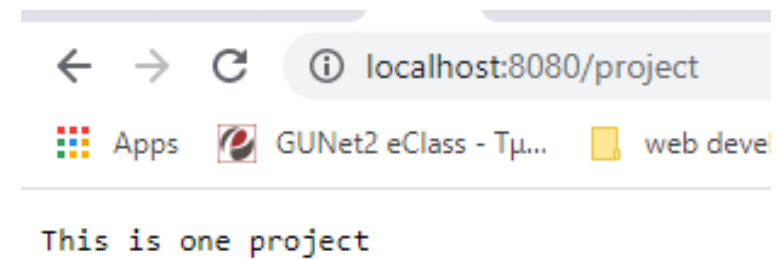
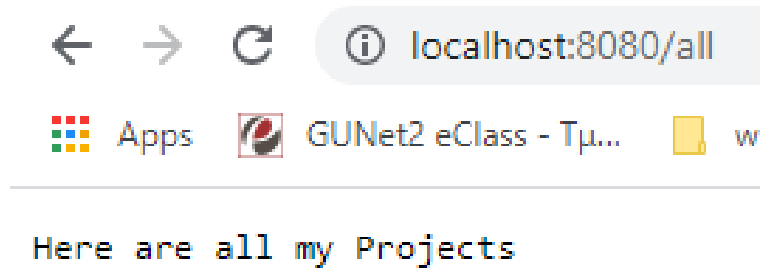
- We use some conditions to handle different urls and display different content
- **req object** represents the **HTTP request** and has **properties**

```
const server= http.createServer((req, res) => {  
  //we store the requested url in a variable  
  const path_name= req.url;  
  console.log(path_name);  
  
  if(path_name=== '/' || path_name=== '/all'){  
    res.write('Here are all my Projects');  
  }else if(path_name=== '/project'){  
    res.write('This is one project');  
  }else{  
    //inspect web page and check the console  
    //we can also send headers, a piece of info regar  
    res.writeHead('404', {'Content-Type': 'text/html'});  
    res.write('<h2> 404 <br> Page not Found </h2>');  
  }  
  res.end(); //end the response  
});
```

200 - OK  
301 - Resource moved  
404 - Not found  
500 - Internal server error

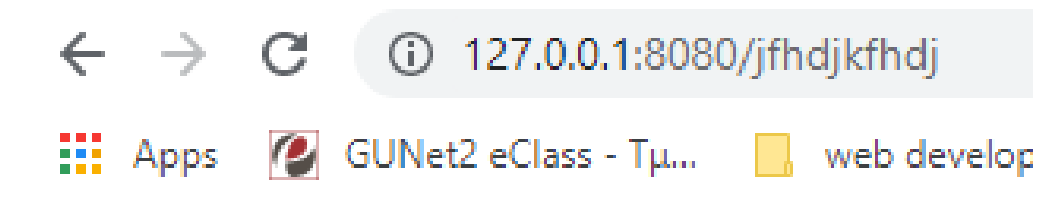
# Lets see an example

- Lets see the results in the browser



# Lets see an example

- Lets see the results in the browser



**404**  
**Page not Found**

- Lets see how we can perform routing and serve an html file

Check

<https://stackoverflow.com/questions/28094192/difference-between-response-setheader-and-response-writehead>

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express Nodejs/deployment](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/deployment)





*To be continued...*