4... Web Server

Objectives...

- To learn about Creating Web Server.
- To study Handling HTTP Request.
- To learn about Sending Request.

4.1 INTRODUCTION [S-22]

- **Web Server:** The word web server comprises of hardware or software, or both of them working simultaneously.
  1. **Hardware Part:** A hardware web server is a computer that stores web server software and a website's component files. (For example, HTML documents, images, CSS stylesheets, and JavaScript files). A web server connects to the Internet and supports request response in terms of information exchange to the devices connected to the web server.
  2. **Software Part:** Software part of web server, manages web users access hosted files on web servers. On current context this is a Hyper Text Transfer Protocol server (HTTP). An HTTP server is software that executes uniform resource locator (web addresses) and HTTP instructions (the protocol browser uses to view HTML web page's). 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.

To publish a website, you need either a Static or Dynamic web server.

1. **Static Web Server:**

- A static web server consists of a computer (hardware) with an HTTP server (software). It is known as static because the server sends its hosted files as it is to browser. Any runtime data processing or execution of instructions cannot be implemented on Static Web Server.
- **Static Website:** Static website is considered as one of the easy website which can be developed in small time. It consists of limited number of pages. Sometime it is of

single page which is like landing page. That page consists of all the information related to your business services.

- **Advantages of Static Websites:**
  1. Convenient to Develop.
  2. Low cost.
  3. Convenient and cheap to host.
- **Disadvantages of Static Websites:**
  1. It requires skilled web developer who can update the site versions.
  2. It requires a strong Search Engine Optimization professional in order to resolve all the technical issues from Google point of view.
  3. Static websites are not end user friendly.

2. Dynamic Web Server:

- A dynamic web server comprises of a static web server and an application server and a database. It is known as dynamic because the application server updates the hosted files before sending content to your browser via the HTTP server.
- **Dynamic Website:** Dynamic websites are known as one of the best website from customer point of view. Dynamic websites are expensive to develop and deploy. Dynamic websites are search engine friendly and easy to update. These website has inbuilt content management system where content can be updated as per need. Now a day's, most of the websites contains dynamic server. For example banking, insurance and airline reservation web applications.
- **Advantages of Dynamic Website:**
  1. Dynamic content development and updating on various static pages of websites.
  2. Very easy to manage and update.
  3. You can update user friendly content related to your services or blogs and rank them in search engines.
- The main **disadvantage** of having dynamic website is they are costly and even their hosting is costly.
- **Static Vs. Dynamic Website:** Following table shows difference between Static Website and Dynamic Website

Table 4.1: Static Vs Dynamic Website

Sr. No.	Static Website	Dynamic Website
1.	When page is loaded the prebuilt content is same every time.	Content is generated quickly and changes regularly.

Contd...

2.	It uses the HTML code for developing a website.	It uses the server side languages such as PHP, SERVLET, JSP, and ASP.NET etc. for developing a website.
3.	It sends exactly the same response for every request.	It may generate different HTML for each of the request.
4.	The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
5.	Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

4.1.1 HTTP Request Response

- On a web server, the HTTP server is responsible for processing and answering incoming requests.
- When browser or a client needs a webpage that is hosted on a web server, the browser requests the file through HTTP. When the request reaches the correct web server (Hardware or IP address), the (software) HTTP server accepts the request, process the requested document, and sends it back to the browser, also through HTTP. (If the server doesn't find the requested document, it returns a 404 response instead.)

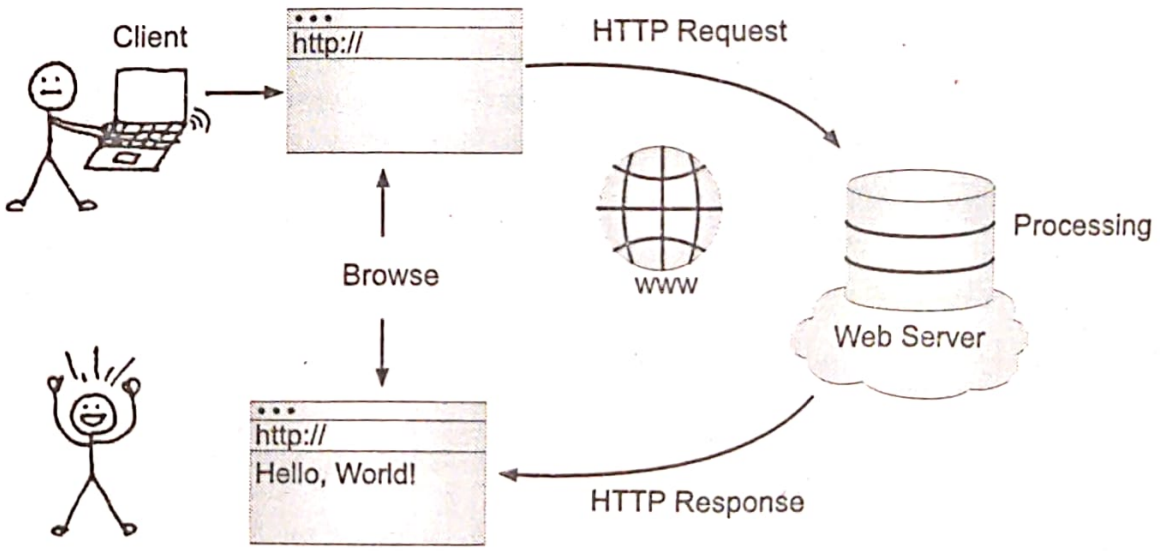


Fig. 4.1: HTTP Request and HTTP Response

4.1.2 Features of HTTP

- **HTTP is Stateless:** HTTP is called as a stateless protocol because there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But

while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

- **HTTP is Connection less:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server know about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.
- **HTTP is Simple:** HTTP is generally designed to be simple and human readable, even with the added complexity introduced in HTTP/2 by encapsulating HTTP messages into frames. HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.
- **HTTP is extensible:** HTTP can be integrated with new functionality by providing a simple agreement between a client and a server.

4.2 CREATING WEB SERVER [W-22, S-23]

- The Node JS Framework is mostly used to create server based applications. The Framework can easily used to create web service which can server content to user.
- There is variety of models such as **http** and **request** module which helps in processing server related request in the web server space. We will have a look at how we can create a basic web server application using node JS.

Creating Node JS Web Server:

- Node JS has a built-in module called HTTP, which allows Node JS to transfer data over the Hyper Text Transfer Protocol (HTTP).
- To include the HTTP module, use the `require()` method:

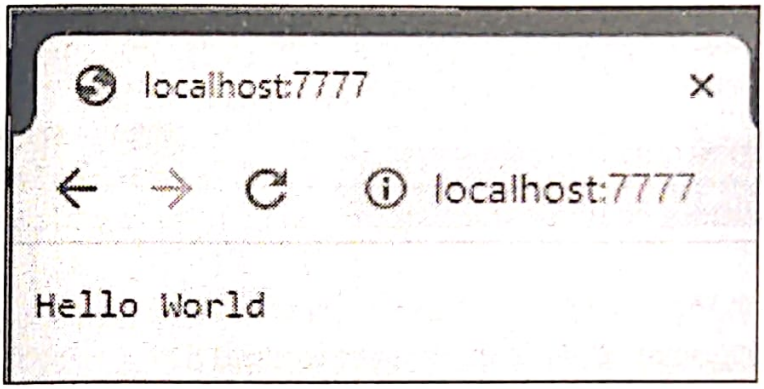
```
var http = require('http');
```

- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- Use the `createServer()` method to create an HTTP server.
- Let's look at an example how to create and run node JS application.
- Our application is going to create a simple server module which will listen on port number 7777 if a request is made through the browser on this port number then server application will send a "Hello World" response to the client.

```
1. var http = require('http')
2. var server = http.createServer((function (request, response) {
```

```
3. response.writeHead(200, { "Content-Type": "text/plain" });
4. response.end("Hello World\n");
5. }));
6. server.listen(7777);
```

Output:



Code explanation:

1. We import the `http` module using `require()` function. The `http` module is a core module of Node JS, so no need to install it using NPM.
2. The next step is to call `createServer()` method of `http` and specify callback function with request and response parameter.
3. When a request is received, we are saying to send a response with the header type of 200 this number is normal response which is sent in an HTTP header when a successful response is sent to the client.
4. In the response itself we are sending the string Hello World.
5. We are then using the `server.listen()` function to make our server application listen to the client request on port number 7777. You can specify any available port over here.

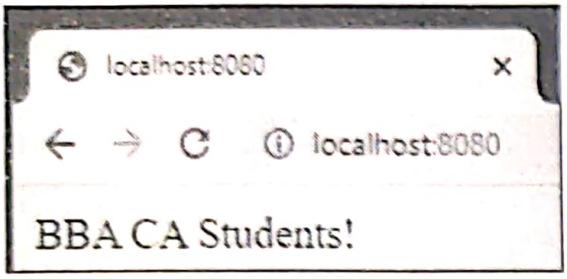
Add an HTTP Header:

- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type.

Program 4.1: Program to add an http header.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('BBA CA Students!');
  res.end();
}).listen(8080);
```

Output:



- The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

4.3 HANDLING HTTP REQUESTS [W-22, S-23]

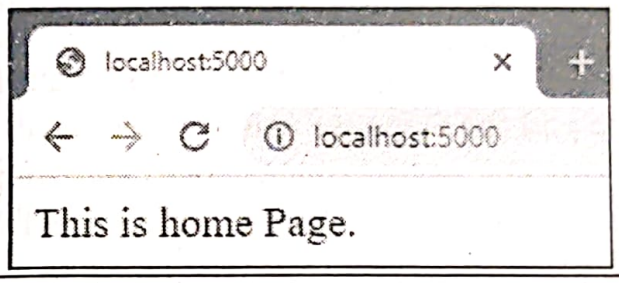
- The `http.createServer()` method includes request and response parameters which is supplied by Node JS.
- The request object can be used to get information about the current HTTP request e.g., URL, request header, and data. The response object can be used to send a response for a current HTTP request.

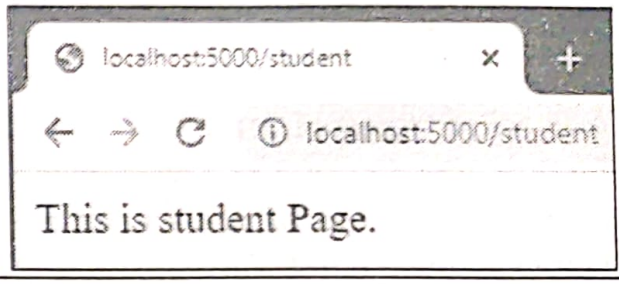
Program 4.2: Program demonstrates handling HTTP request and response in Node JS.

```
var http = require('http');

var server = http.createServer(function (req, res) {
    if (req.url == '/') {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is home Page.</p></body></html>');
       res.end();
    }

   else if (req.url == "/student") {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is student Page.</p></body></html>');
       res.end();
    }

   else if (req.url == "/admin") {
       res.writeHead(200, { 'Content-Type': 'text/html' });
       res.write('<html><body><p>This is admin Page.</p></body></html>');
       res.end();
    }

   else
```

```
        res.end('Invalid Request!');
});

server.listen(5000);

console.log('Node JS  web server at port 5000 is running..')
```

Output:





- The above example, `req.url` is used to check the URL of the current request and based on that it sends the response. To send a response, first it sets the response header using `writeHead()` method and then writes a string as a response body using `write()` method. Finally, Node JS web server sends the response using `end()` method.
- Now, run the above web server as shown below.

```
PS D:\Prajakta\BBA CA\Node JS\programs> node app.js
Node JS  web server at port 5000 is running.
```

- **1st Output Screen:** For Windows users, point your browser to `http://localhost:5000` and see the 1st output screen.
- **2nd Output Screen:** The same way, point your browser to `http://localhost:5000 student` and see 2nd output screen.

4.3.1 How Node Presents Incoming HTTP Requests to Developers

- Node provides HTTP server and client interfaces through the http module:

```
var http = require('http');
```

- To create an HTTP server, call the `http.createServer()` function. It accepts a single argument, a callback function that will be called on each HTTP request received by the server. This request callback receives as arguments, the request and response objects, which are commonly shortened to `req` and `res`:

```
var http = require('http');
var server = http.createServer(function(req, res){
// handle request
});
```

- For every HTTP request received by the server, the request callback function will be invoked with new `req` and `res` objects. Prior to the callback being triggered, Node will parse the request up through the HTTP headers and provide them as part of the `req` object. But Node doesn't start parsing the body of the request until the callback has been fired. This is different from some server side frameworks, like PHP, where both the headers and the body of the request are parsed before your application logic runs. Node provides this lower level interface so you can handle the body data as it is being parsed, if desired.
- Node will not automatically write any response back to the client. After the request callback is triggered, it's your responsibility to end the response using the `res.end()` method (see figure 4.2). This allows you to run any asynchronous logic you want during the lifetime of the request before ending the response. If you fail to end the response, the request will hang until the client times out or it will just remain open.

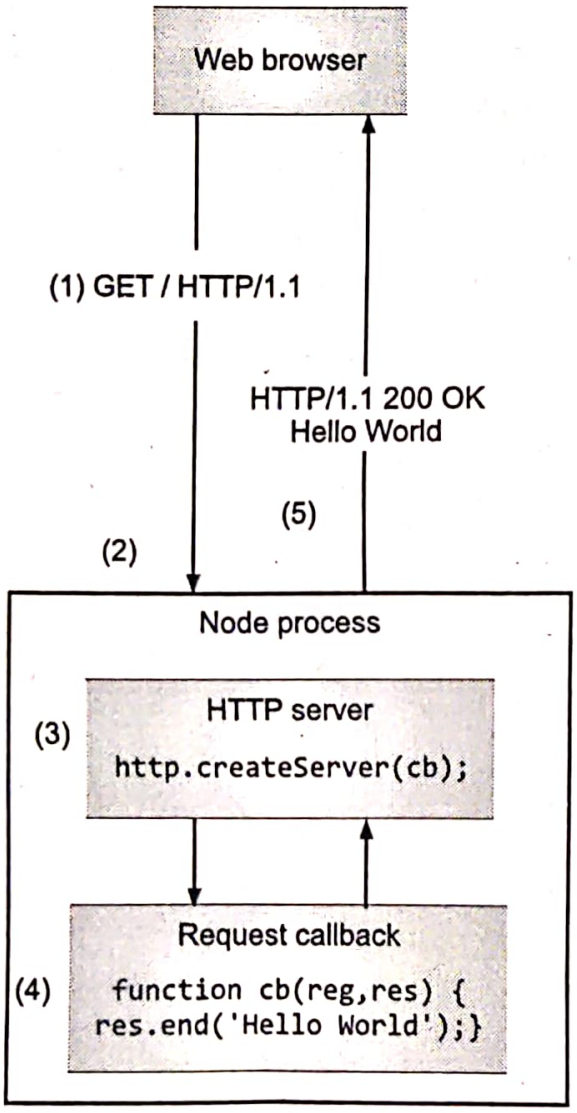


Fig. 4.2: The lifecycle of an HTTP request going through a Node HTTP server

(1) An HTTP client like a web browser initiates an HTTP request.

(2) Node accepts the connection, and incoming request data is given to the HTTP server.

(3) The HTTP server parses up to the end of the HTTP headers and then hands control over to the request callback.

(4) The request callback performs application logic, in this case responding immediately with the text "Hello World".

(5) The request is sent back through the HTTP server, which formats a proper HTTP response for the client.

- Node servers are long-running processes that serve many requests throughout their lifetimes.

Reading Request Headers and Setting Response Headers:

- The Hello World example in the previous section demonstrates the bare minimum required for a proper HTTP response. It uses the default status code of 200 (indicating success) and the default response headers. Usually, though, you'll want to include any number of other HTTP headers with the response. For example, you'll have to send a Content-Type header with a value of text/html when you're sending HTML content so that the browser knows to render the result as HTML.
- Node offers several methods to progressively alter the header fields of an, HTTP response: `res.setHeader(field, value)`, `res.getHeader(field)`, `res.removeHeader(field)` methods. Here's an example of using `res.setHeader()`:

```
var body = 'Hello World'; res.setHeader('Content-Length', body.length);
res.setHeader('Content-Type', 'text/plain'); res.end(body);
```

- You can add and remove headers in any order, but only up to the first `res.write()` or `res.end()` call. After the first part of the response body is written, Node will flush the HTTP headers that have been set.

4.4 SENDING REQUESTS

- The `req` parameter of the `http.createServer()` method has all the information about the incoming request for eg. request payload, headers, URL etc.
- The `req` parameter has the following properties:
  1. **request.headers:** Information about the request headers such as Connection, Host, Authorization, etc.
  2. **request.method:** Information about the incoming request methods such as GET, POST, PUT, DELETE, OPTIONS, HEAD, etc.
  3. **request.url:** Information about the incoming request URL, such as /accounts, /users, /messages etc.

Request Method:

a. GET

- Most requests are GET requests Node JS provides this method. The only difference between this method and `http.request()` is that it sets the method to GET and calls `req.end()` automatically.
- **Example:**

```
const https = require('https')
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'GET'
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})
req.end()
```

b. POST

- `http.request()` returns instance `http.ClientRequest` class.
- The ClientRequest instance is a writable stream. If one needs to upload a file with a POST request, then write to the ClientRequest object.
- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

c. PUT

PUT requests use the same POST request format and just change the `options` method value.

- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

d. DELETE

DELETE requests use the same POST request format and just change the `options` method value.

- **Example:**

```
const https = require('https')

const data = JSON.stringify({
  todo: 'Buy the milk'
})
```

```
const options = {
  hostname: 'anyURL.com',
  port: 450,
  path: '/myfolder',
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': data.length
  }
}

const req = https.request(options, res => {
console.log('statusCode: ${res.statusCode}')

res.on('data', d => {
process.stdout.write(d)
  })
})

req.on('error', error => {
console.error(error)
})

req.write(data)
req.end()
```

e. HEAD

- The response to an HTTP HEAD request is always just the response header. The response body is always 0 length.
- **Example:**

```
'use strict';

const http = require('http');

http.request('http://example.com', { method: 'HEAD' }, (res) => {
  console.log(res.statusCode);
}).on('error', (err) => {
console.error(err);
}).end();
```

Summary

- **Web Server:** The word web server comprises of hardware or software, or both of them working simultaneously.
- **Hardware Part:** A hardware web server is a computer that stores web server software and a website's component files.
- **Software Part:** Software part of web server, manages web users access hosted files on web servers. On current context this is a Hyper Text Transfer Protocol server (HTTP).
- A static web server consists of a computer (hardware) with an HTTP server (software). It is known as static because the server sends its hosted files as it is to browser.
- Static website is considered as one of the easy website which can be developed in small time. It consists of limited number of pages.
- A dynamic web server comprises of a static web server and an application server and a database. It is known as dynamic because the application server updates the hosted files before sending content to your browser via the HTTP server.
- Dynamic web sites are known as one of the best website from customer point of view. Dynamic websites are expensive to develop and deploy. Dynamic websites are search engine friendly and easy to update.
- There is variety of models such as http and request module which helps in processing server related request in the web server space.

Check Your Understanding

1. The word web server comprises which of the following?
   (a) hardware (b) software
   (c) Both (d) None of the above
2. Which is not advantage of static websites?
   (a) Convenient to develop (b) Low cost.
   (c) Convenient and cheap to host. (d) High cost
3. HTTP is _____.
   (a) Stateless (b) Stateful
   (c) Both the above (d) None of the above
4. Which method includes request and response parameters which are supplied by Node JS?
   (a) http.createServer()
   (b) http.MakeServer()
   (c) http.GetServer()
   (d) None of the above

5. Difference between GET method and http.request() is_____.
   (a) It sets the method to GET and calls req.end() automatically.
   (b) It sets the method to GET and calls req.terminate() automatically.
   (c) It sets the method to GET and calls req.finish() automatically
   (d) None of the above
6. HTTP is_____.
   (a) Connection less
   (b) Connection oriented
   (c) Both a & b
   (d) None of the above
7. PUT, POST and which requests use the same request format and just change the options method value.
   (a) DELETE
   (b) GET
   (c) HEAD
   (d) None of the above
8. The main disadvantages of having dynamic website is _____.
   (a) They are costly and even their hosting is costly.
   (b) Less costly
   (c) Hosting is less costly
   (d) None of the above

ANSWER KEY

1. (c)	2. (d)	3. (a)	4. (a)	5. (a)
6. (a)	7. (a)	8. (a)		

Practice Questions

Q.I Answer the following questions in short.

1. List out sending request methods.
2. What is difference between GET and http.request() method?
3. Explain working of writeHead().
4. Explain advantages and disadvantages of static website.
5. Explain advantages and disadvantages of dynamic website.

Q.II Answer the following questions.

1. Explain web server with real example?
2. Write difference between static website and dynamic website.

3. How to read query string, write sample code and explain it?
4. How to split query string, take example of splitting first name and last name from given user name?
5. Explain the function of requestListener() function with suitable example?
6. How to read text from web pages?
7. Write Steps to create web Server?
8. What are functions of HTTP?

Q.III Define the terms.

1. Web site
2. Web Server
3. Static website
4. Dynamic website