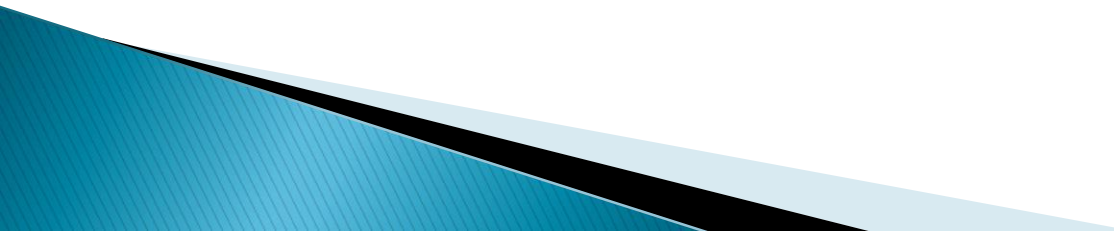


# Node.JS

## Lecture 5

### HTTP Server, URL and Node-static

# Road Map

- ▶ HTTP Module
    - Request Object
    - Response Object
  - ▶ URL
  - ▶ Route
  - ▶ Node-Static
  - ▶ Coding Practice
- 

# HTTP & URL Module

# HTTP

- ▶ The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- ▶ The server executes the `createServer()` callback each time a request is received from a client.
  - The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on specified port.
  - `close()` stops the server from accepting new connections
  - `listen()` starts the HTTP server and listens for connections

# HTTP Request

- ▶ The “request” argument contains information about the header, method(Verb) and the requested resource(URL).
- ▶ Method is Post, Get, Patch and put etc
- ▶ The Request object has a property called "url" which holds the part of the url that comes after the domain name. (req.url)
  - The url is the full URL without the server, protocol or port.
- ▶ Req.header will provide you header information

# HTTP Methods possible

'ACL',  
'BIND',  
'CHECKOUT',  
'CONNECT',  
'COPY',  
'DELETE',  
'GET',  
'HEAD',  
'LINK',  
'LOCK',  
'M-SEARCH',  
'MERGE',  
'MKACTIVITY',  
'MKCALENDAR',  
'MKCOL',

'MOVE',  
'NOTIFY',  
'OPTIONS',  
'PATCH',  
'POST',  
'PROPFIND',  
'PROPPATCH',  
'PURGE',  
'PUT',  
'REBIND',  
'REPORT',  
'SEARCH',  
'SUBSCRIBE',  
'TRACE',  
'UNBIND',  
'UNLINK',  
'UNLOCK',  
'UNSUBSCRIBE' ]

# HTTP Response

- ▶ The “response” argument, on the other hand, contains information regarding server response.
- ▶ 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:
- ▶ The `writeHead()` method is used to write the status code and any optional response headers(Mime Type).

```
res.writeHead(200,{'Content-Type': 'text/html'});
```

# HTTP Response

- ▶ Data is written using zero or more `res.write()` calls.
- ▶ After all the data has been written, the server completes the connection with a single **mandatory** call to `res.end()`.
- ▶ `res.end()` supports the same arguments as `res.write()`.
- ▶ This means that the entire response can be written via `res.end()`
- ▶ If you want to send multiline response, you can use back tick “`” at start and at end of string.

```
res.end(`
    <!doctype html>
    <html>
    <body>
        <form action="/" method="post">
            <input type="text" name="fname" /><br />
        <button>Save</button>
        </form>
    </body>
    </html>`);
```



# Example

This basic server sends back a single response header, Content-Type, a 200 response status code, and the body, Hello World\n:

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.end('Hello World\n');  
}).listen(1337, '127.0.0.1');  
console.log('Server running at http://127.0.0.1:1337/');
```

# Status Codes

'100': 'Continue',  
'101': 'Switching Protocols',  
'102': 'Processing',  
'200': 'OK',  
'201': 'Created',  
'202': 'Accepted',  
'203': 'Non-Authoritative  
Information',  
'204': 'No Content',  
'205': 'Reset Content',  
'206': 'Partial Content',  
'207': 'Multi-Status',  
'208': 'Already Reported',  
'226': 'IM Used',  
'300': 'Multiple Choices',

'301': 'Moved Permanently',  
'302': 'Found',  
'303': 'See Other',  
'304': 'Not Modified',  
'305': 'Use Proxy',  
'307': 'Temporary Redirect',  
'308': 'Permanent Redirect',  
'400': 'Bad Request',  
'401': 'Unauthorized',  
'402': 'Payment Required',  
'403': 'Forbidden',  
'404': 'Not Found',  
'405': 'Method Not Allowed',  
'406': 'Not Acceptable',  
'407': 'Proxy Authentication Required',  
'408': 'Request Timeout',  
'409': 'Conflict',  
'410': 'Gone',

# Status Codes

'411': 'Length Required',

'412': 'Precondition Failed',

'413': 'Payload Too Large',

'414': 'URI Too Long',

'415': 'Unsupported Media  
Type',

'416': 'Range Not Satisfiable',

'417': 'Expectation Failed',

'418': 'I\'m a teapot',

'421': 'Misdirected Request',

'422': 'Unprocessable Entity',

'423': 'Locked',

'424': 'Failed Dependency',

'425': 'Unordered Collection',

'426': 'Upgrade Required',

'428': 'Precondition Required',

'429': 'Too Many Requests',

'431': 'Request Header Fields Too Large',

'451': 'Unavailable For Legal Reasons',

'500': 'Internal Server Error',

'501': 'Not Implemented',

'502': 'Bad Gateway',

'503': 'Service Unavailable',

'504': 'Gateway Timeout',

'505': 'HTTP Version Not Supported',

'506': 'Variant Also Negotiates',

'507': 'Insufficient Storage',


'508': 'Loop Detected',

'509': 'Bandwidth Limit Exceeded',

'510': 'Not Extended',

'511': 'Network Authentication Required'

# URL Module

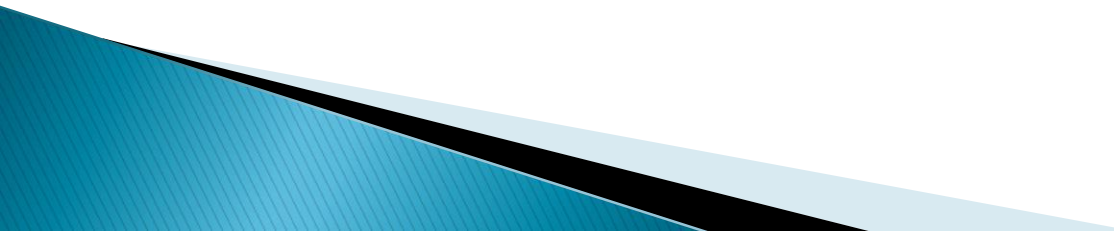
- ▶ The URL module splits up a web address into readable parts.
  - ▶ Parse an address with the `url.parse()` method, and it will return a URL object with each part of the address as properties as seen in next slide picture.
  - ▶ URL Module is applied normally on request received in a Node. JS server.
- 

href						
protocol	auth		host		path	hash
			hostname	port	pathname	search
						query
"	https:	//	user	:	pass	@ sub.host.com : 8080 /p/a/t/h ? query=string #hash "
			hostname	port		
protocol	username	password	host			
origin			origin	pathname	search	hash
href						

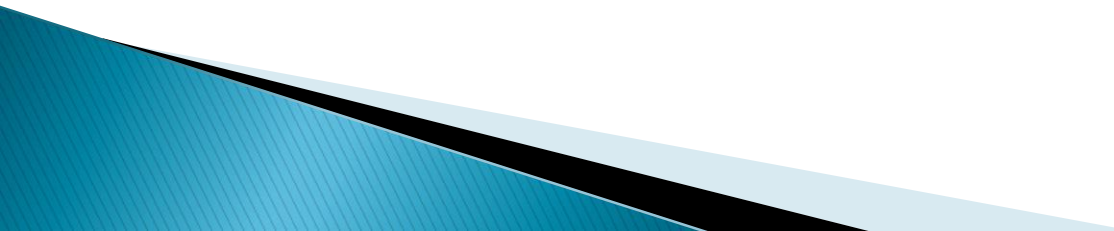
(all spaces in the "" line should be ignored – they are purely for formatting)

# url.parse()

```
var url = require('url');  
var adr= 'http://localhost:8080/default.htm?year=2017&month  
=february';  
var q = url.parse(adr);  
console.log(q.host); //returns 'localhost:8080'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?year=2017&month=february'  
var qdata = q.query; //returns an object: { year: 2017, month:  
'february' }  
console.log(qdata.month); //returns 'february'
```



# Routes

- ▶ The combination of an HTTP verb(request method) and requested URL is known as a **route**.
  - ▶ Notice that the requested URL is available via `req.url`, while the HTTP verb is found in `req.method`
  - ▶ We will use this route to get request and write response accordingly.
- 

# example

```
var http = require('http');
http.createServer(function(req, res) {
  if (req.url === '/' && req.method === 'GET') {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('Hello <strong>home page</strong>');
  } else if (req.url === '/account' && req.method === 'GET') {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end("Hello <strong>account page</strong>");
  } else {
    res.writeHead(404, {'Content-Type': 'text/html'});
    res.end();
  }
}).listen(1337);
```



```
var http = require('http');
var fs = require('fs');
var url = require('url');
// Create a server
http.createServer( function (request,
response) {
// Parse the request containing file name
    var pathname =
url.parse(request.url).pathname;

// Print the name of the file for which
request is made.
    console.log("Request for " + pathname +
" received.");

// Read the requested file content from
file system
    fs.readFile(pathname.substr(1), function
(err, data) {
```

```
if (err) {
    console.log(err);
    // HTTP Status: 404 : NOT FOUND
    // Content Type: text/plain
    response.writeHead(404, {'Content-Type':
'text/html'});
} else {
    //Page found
    // HTTP Status: 200 : OK
    // Content Type: text/plain
    response.writeHead(200, {'Content-Type':
'text/html'});

// Write the content of the file to response body
    response.write(data.toString());

}
// Send the response body
    response.end();
});
}).listen(8081);

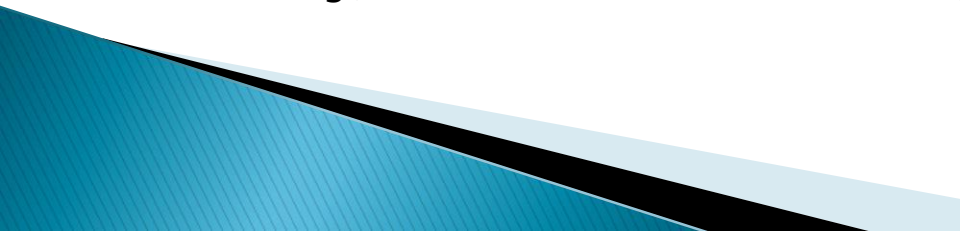
// Console will print the message
console.log('Server running at
http://127.0.0.1:8081/');
```

# Request Body

- ▶ To access data sent from the form, you need event emitter on req object

```
if (req.method === 'POST') {  
  let body = "";  
  req.on('data', chunk => { // chunk is default stream data set  
    body += chunk.toString(); // convert Buffer to string  
  });  
  req.on('end', () => {  
    console.log(body);  
    res.end('ok');  
  });  
}
```

```
var http = require('http');
http.createServer(function(req,res){
  // normalize url by removing querystring, optional // trailing slash, and making it lowercase
  var path = req.url.replace(/\/?(\?\.*)?$/, "").toLowerCase();
  switch(path) {
    case '':
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('Homepage');
      break;
    case '/about':
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('About');
      break;
    default:
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end('Not Found');
      break;
  }
}).listen(3000);
console.log('Server started on localhost:3000; press Ctrl-C to terminate....');
```



# Node Static

A Sample External Module

# Node-static

```
var static = require( 'node-static' ),
    port = 3000,
    http = require( 'http' );
// config
var file = new static.Server( './public', {
    cache: 3600,
    gzip: true
} );
// serve
http.createServer( function ( request, response ) {
    request.addListener( 'end', function () {
        file.serve( request, response );
    } ).resume();
} ).listen( port )
```

# Static.server

- ▶ This will serve files in the current directory. If you want to serve files in a specific directory, pass it as the first argument:
- ▶ `new static.Server('./public');`
  - You can also specify how long the client is supposed to cache the files node-static serves:
- ▶ `new static.Server('./public', { cache: 3600 });`
  - This will set the Cache-Control header, telling clients to cache the file for an hour. This is the default setting.

# Static.server()

## ▶ gzip

- Enable support for sending compressed responses. This will enable a check for a file with the same name plus '.gz' in the same folder.
- If the compressed file is found and the client has indicated support for gzip file transfer, the contents of the .gz file will be sent in place of the uncompressed file along with a Content-Encoding: gzip header to inform the client the data has been compressed.
- example: { gzip: true } example: { gzip: /^\/text/ }
- Passing true will enable this check for all files. Passing a RegExp instance will only enable this check if the content-type of the respond would match that RegExp using its test() method.
- Defaults to false

```
request.addListener( 'end', function () {  
    file.serve( request, response );  
} )
```

- ▶ This will add an event listener on request.
- ▶ When request.end is called, this callback function will be called which will serve file as response.