Introduction To Full-Stack Web Development

CS 386

Michael Kremer

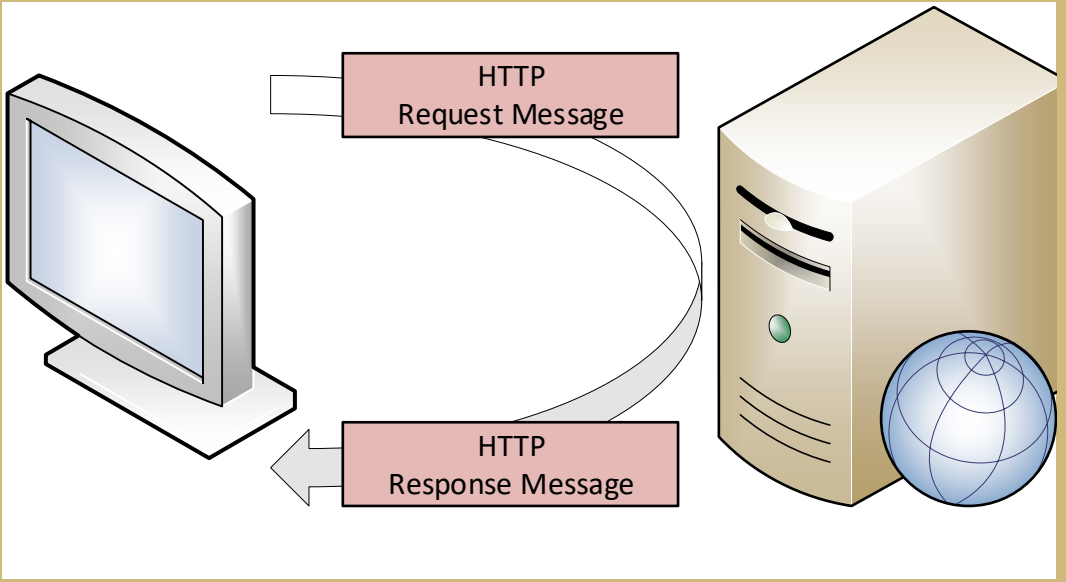
Last updated: 11/8/2023 8:26:50 AM

Class 21

- 21.1 HTTP Protocol
- 21.2 Request Object
- 21.3 Response Object

21.1 HTTP Protocol

- HTTP (Hypertext Transfer Protocol) is perhaps most popular application protocol used in Internet (or The WEB).
- HTTP is asymmetric request-response client-server protocol:
  - HTTP client sends request message to HTTP server
  - Server returns response message
  - In other words:
    - HTTP is pull protocol
  - Client pulls information from server (instead of server pushes information down to client)



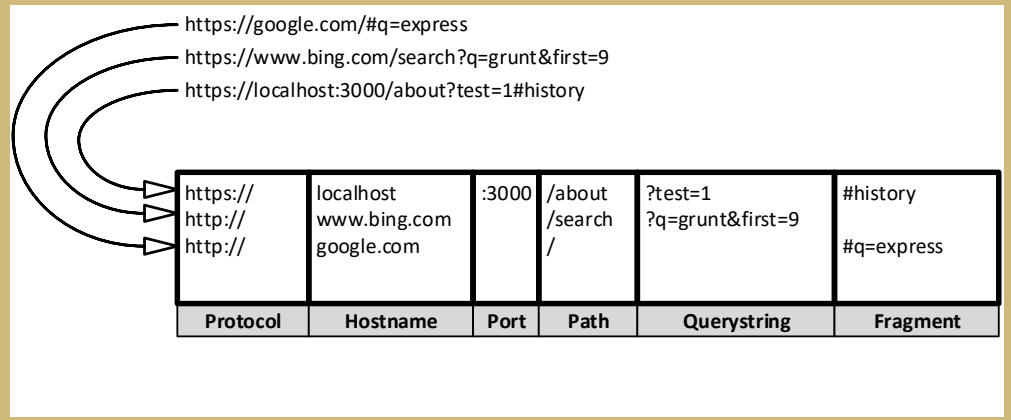
21.1 HTTP Protocol

- Like most Internet protocols:
  - It is command and response text-based protocol
  - Using client server communications model
- HTTP protocol is also stateless protocol:
  - Server is not required to store session information
  - Each request is independent of each other
- All requests originate at client (your browser)
  - Server responds to request
  - Requests (commands) and responses are in readable text
  - Requests are independent of each other and server does not need to track requests (stateless)

21.1 HTTP Protocol

URL (Uniform Resource Locator)

- URL in browser's address bar contains different parts that have specific meanings
- **Protocol:**
  - Determines how request will be transmitted (http, https, ftps)
- **Host:**
  - Identifies the server
  - Servers on your computer (localhost) or local network may simply be one word, or it may be numeric IP address
  - On Internet, host will end in top-level domain (TLD) like .com or .net
  - Additionally, there may be subdomains, which prefixes hostname
  - www is very common subdomain, though it can be anything (Subdomains are optional)



21.1 HTTP Protocol

URL (Uniform Resource Locator)

- **Port:**
  - Each server has collection of numbered ports
  - Some port numbers are "special," like 80 and 443
  - When port is omitted, port 80 is assumed for HTTP and 443 for HTTPS
  - In general, if not using port 80 or 443, use port number greater than 1023
  - Very common to use easy-to-remember port numbers like 3000, 8080, and 8088
- **Notes:**
  - Ports with numbers 0 –1023 are called system or well-known ports
  - Ports with numbers 1024 - 49151 are called user or registered ports
  - Ports with numbers 49152 - 65535 are called dynamic and/or private ports
  - Both system and user ports are used by transport protocols (TCP, UDP, DCCP, SCTP) to indicate application or service

21.1 HTTP Protocol

URL (Uniform Resource Locator)

- **Path:**
  - Path is generally first part of URL that your app cares about
  - Possible to make decisions based on protocol, host, and port, but not good practice
  - Path should be used to uniquely identify pages or other resources in your app
- **Querystring:**
  - Querystring is optional collection of name/value pairs
  - Querystring starts with question mark (?)
  - Then name/value pairs separated by ampersands (&)
  - Both names and values should be URL encoded:
    - JavaScript provides built-in function to do that: encodeURIComponent
  - Example:
    - Spaces will be replaced with plus signs (+)
    - Other special characters will be replaced with numeric character references

21.1 HTTP Protocol

URL (Uniform Resource Locator)

- **Fragment:**
  - Fragment (or hash) is not passed to server at all: it is strictly for use by browser
  - Common for single-page/AJAX heavy applications to use fragment to control application
  - Originally, fragment's sole purpose was to cause browser to display specific part of the document, marked by anchor tag (<a id="chapter06">)
- Nodejs has core module url to handle all parts of given url
- url object contains properties and methods:

Property	Description
href	The full URL that was originally parsed. Both the protocol and host are lowercased
protocol	The request protocol, lowercased
host	The full lowercased host portion of the URL, including port information
auth	The authentication information portion of a URL
hostname	Just the lowercased hostname portion of the host
port	The port number portion of the host
pathname	The path section of the URL, that comes after the host and before the query, including the initial slash if present
search	The 'query string' portion of the URL, including the leading question mark
path	Concatenation of pathname and search
query	Either the 'params' portion of the query string, or a querystring-parsed object
hash	The 'fragment' portion of the URL including the pound-sign

21.1 HTTP Protocol

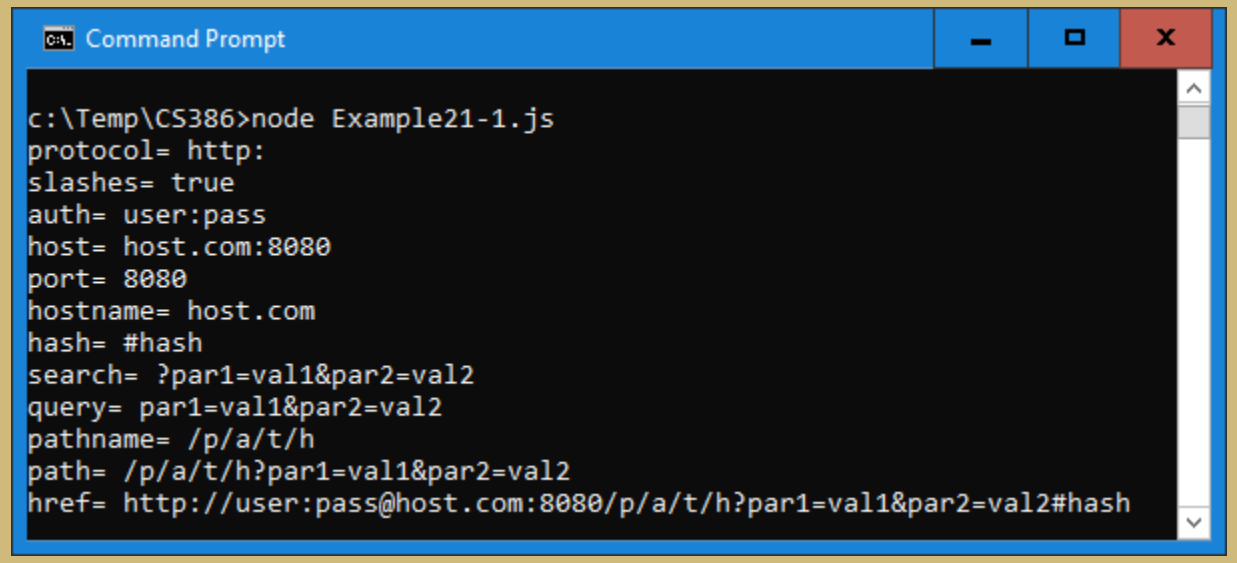
NodeJS core module url

- parse method
  - urlString: It holds the URL string which needs to parse
  - parseQueryString (Boolean, default false):
    - If true then query property will be set to object returned by querystring module's parse() method
    - If it set to false then the query property on the returned URL object will be an unparsed, undecoded string
  - slashesDenoteHost (Boolean, default false) : If true then first token after literal string // and preceding next / will be interpreted as host

Syntax:
url.parse(*urlString* [**,** *parseQueryString*]**,** [*slashesDenoteHost*]**)**

21.1 HTTP Protocol

- **Example 21-1:**
- Load core module url
- Create variable sampleURL, assign following string:
  - http://user:pass@host.com:8080/p/a/t/h?par1=val1&par2=val2#hash
  - Create object variable objURL by using url parse method
- Loop over objURL using iteration variable prop
- Exclude methods:
  - typeof(objURL[prop]) !== 'function')
- Display the property name and value separated by =



21.1 HTTP Protocol

➢ **Example 21-1:**

C:\Temp\CS386\Example21-1.js - Notepad++

```
const url = require('url'); //Load url core module
const sampleURL = 'http://user:pass@host.com:8080/p/a/t/h?par1=val1&par2=val2#hash'; //Sample url
const objURL = url.parse(sampleURL); //Create url object
for (let prop in objURL) { //Loop over url object
    if (typeof(objURL[prop]) !== 'function') { //E
        console.log(prop + "= " + objURL[prop]); /
    }
}
```

JavaScript file | length : 406 lines : 11 | Ln : 11 Col : 1 Pos : 4

Command Prompt

```
c:\Temp\CS386>node Example21-1.js
protocol= http:
slashes= true
auth= user:pass
host= host.com:8080
port= 8080
hostname= host.com
hash= #hash
search= ?par1=val1&par2=val2
query= par1=val1&par2=val2
pathname= /p/a/t/h
path= /p/a/t/h?par1=val1&par2=val2
href= http://user:pass@host.com:8080/p/a/t/h?par1=val1&par2=val2#hash
```

21.1 HTTP Protocol

- **Example 21-1 (continued):**
- Convert query string into object by passing true as second argument
- Using query property to only display query string

C:\Temp\CS386\Example21-1.js - Notepad++

```
const url = require('url'); //Load url core module
const sampleURL = 'http://user:pass@host.com:8080/p/a/t/h?par1=val1&par2=val2#hash'; //Sample url
//const objURL = url.parse(sampleURL); //Create url object
const objURL = url.parse(sampleURL, true).query; //Create query string object
for (let prop in objURL) { //Loop over url object
    if (typeof(objURL[prop]) !== 'function') { //Exclude functions
        console.log(prop + "= " + objURL[prop]); //Property name and value
    }
```

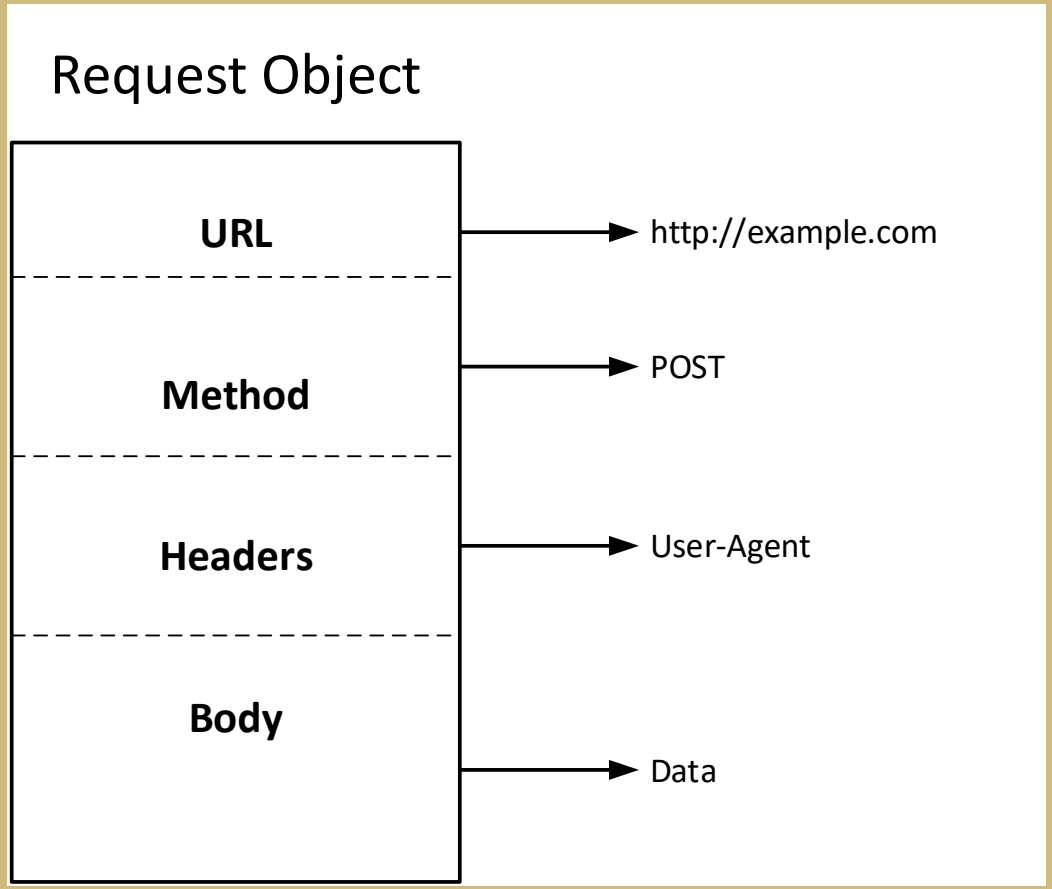
JavaScript file | length : 487 lines : 12 | Ln : 12 Col : 1 Pos : 488 | Windows (CR LF) | UTF-8 | INS

Command Prompt

```
c:\Temp\CS386>node Example21-1.js
par1= val1
par2= val2
```

21.2 Request Object

- Request object (normally passed to callback)
- Can name it whatever you want:
  - Common to name it req or request
  - Starts its life as instance of http.IncomingMessage (core Node object)
- Most important properties and methods of request object



21.2 Request Object

- HTTP protocol defines collection of request methods (often referred to as HTTP verbs) that client uses to communicate with server
- By far, most common methods are GET and POST
- When URL is typed into browser (or click link):
  - Browser issues HTTP GET request to server
- Important information passed to server:
  - URL path and querystring
- Combination of method, path, and querystring is what app uses to determine how to respond
- For static websites, most pages will respond to GET requests
- POST requests are usually reserved for sending information back to server (form processing, for example)

21.2 Request Object

Request Headers:

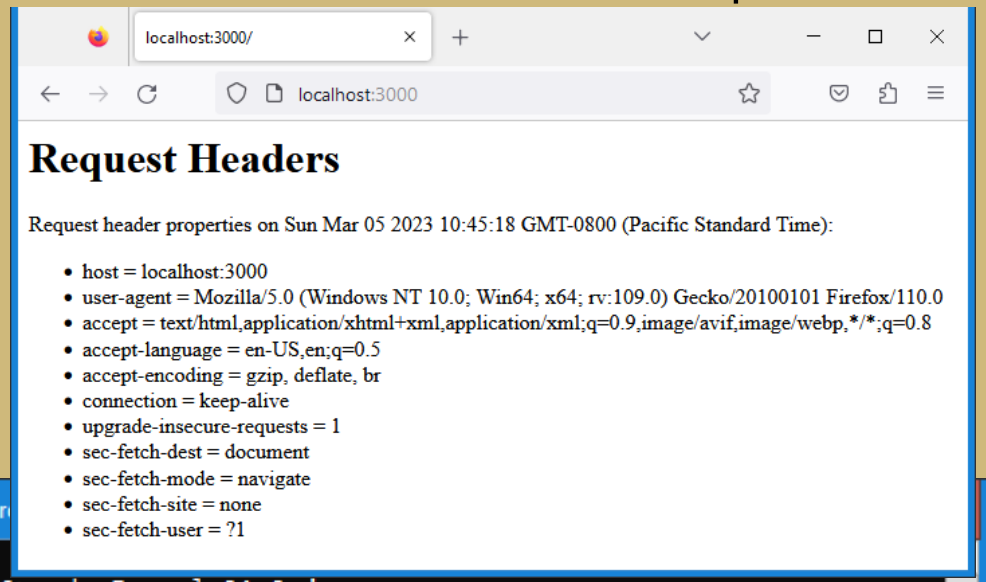
- URL is not only thing that is passed to server when navigating to specific page
- Browser is sending lot of "invisible" information every time website is visited
- Browser will tell server what language it prefers to receive page
- Example:
  - To download Chrome in Spain, it will request Spanish version of visited pages, if they exist
- Also sends information about "user agent" (browser, operating system, and hardware) and other bits of information
- All this information is sent as request header, which is made available through request object's headers property

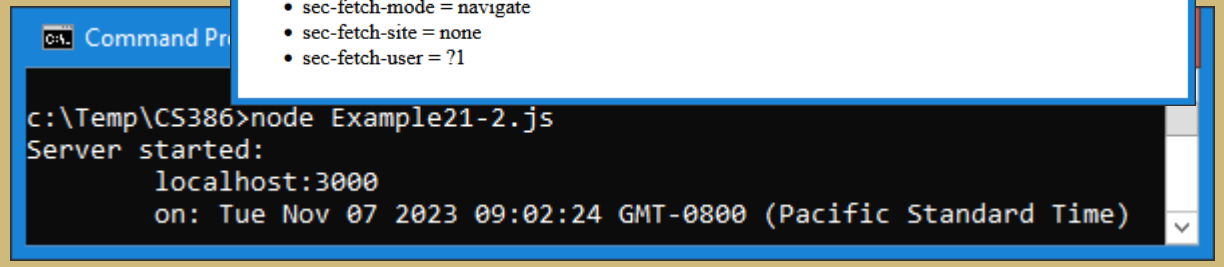
Syntax:

```
const server = https.createServer( function(req, res) {
    let reqHeaders = req.headers;
}
```

21.2 Request Object

- **Example 21-2:**
- Create constant http loading http module
- Create const port assigning 3000, host assigning localhost
- Create const server, assign createServer method with callback function(req, res):
  - Create const reqHeader, assign headers property or request object (req)
  - Loop over headers and accumulate string strHeaders, wrapping each header item into <li> element
  - In end method of response object return string:
    - html string as shown
    - Wrap strHeaders inside
- Listen to server connections:





21.2 Request Object

➢ **Example 21-2:**

C:\Temp\CS386\Example21-2.js - Notepad++

```javascript
const http = require('http');//Load http module
const port = 3000, host = 'localhost'; //Define host, port
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into server constant
    if (req.method === 'GET'){//Check for request method
        const reqHeaders = req.headers;//Store request headers object in variable
        let strHeaders = '';//Declare string variable
        for (let prop in reqHeaders) {//Loop over all properties p in request header object
            strHeaders += "<li>" + prop + " = " + reqHeaders[prop] + "</li>";//Build string of li elements
        }//Ending for loop curly brace
        //Sending response
        res.end(`<h1>Request Headers</h1>
        <p>Request header properties on ${new Date()}:</p>
        <ul> ${strHeaders} </ul>`);//Merging strHeaders into html markup
    };//Ending if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for incor
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,201 lines : 25 | Ln : 25 Col : 1 Pos : 1,202

localhost:3000/

localhost:3000

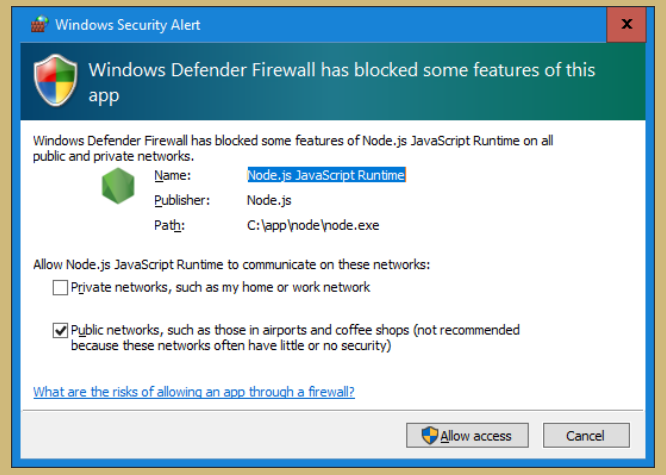
Request Headers

Request header properties on Sun Mar 05 2023 10:45:18 GMT-0800 (Pacific Standard Time):

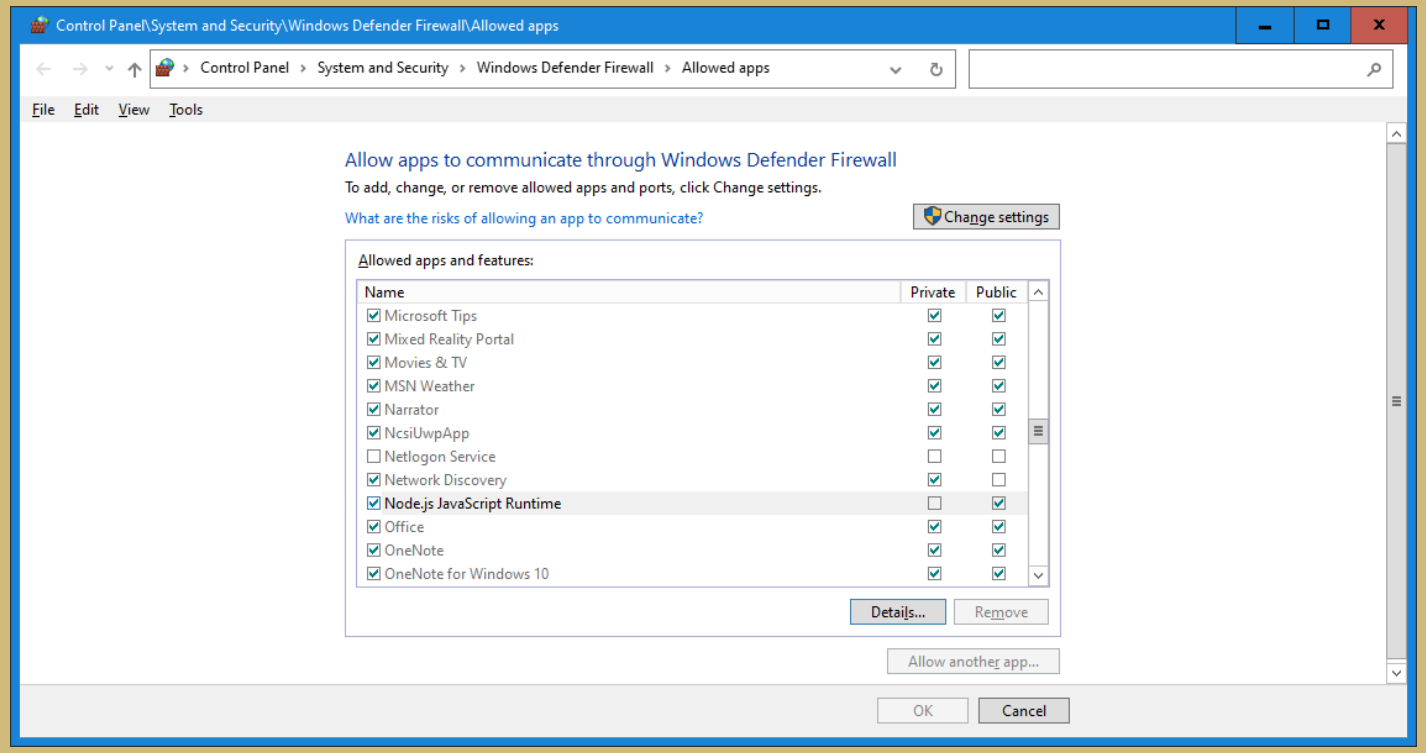
- host = localhost:3000
- user-agent = Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0
- accept = text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
- accept-language = en-US,en;q=0.5
- accept-encoding = gzip, deflate, br
- connection = keep-alive
- upgrade-insecure-requests = 1
- sec-fetch-dest = document
- sec-fetch-mode = navigate
- sec-fetch-site = none
- sec-fetch-user = ?1

21.2 Request Object

- **Example 21-2:**
- Might see Firewall warning message when running



- Click on Allow access



21.2 Request Object

Request URL/OriginalUrl

- req.url returns path and querystring (excluding protocol, host, or port)
- req.url can be rewritten for internal routing purposes
- But req.originalUrl is designed to remain original request and querystring
- Use core module url to easily parse url into its components
- See previous example (21-1)

21.2 Request Object

Request Body

- In addition to request headers, requests can have body:
  - Similar to body of response which is actual content being returned
- Normal GET requests do not have bodies, but POST requests usually do (data)
- Most common media type for POST bodies is application/x-www-form-urlencoded
- Simply encoded name/value pairs separated by ampersands (essentially same format as querystring)
- If POST needs to support file uploads → media type is multipart/form-data
- Lastly, AJAX requests can use application/json for body

➢ For next Examples, download zip file from Canvas

21.2 Request Object

- **Example 21-3:**
- Complete following steps in Example21-3.js:

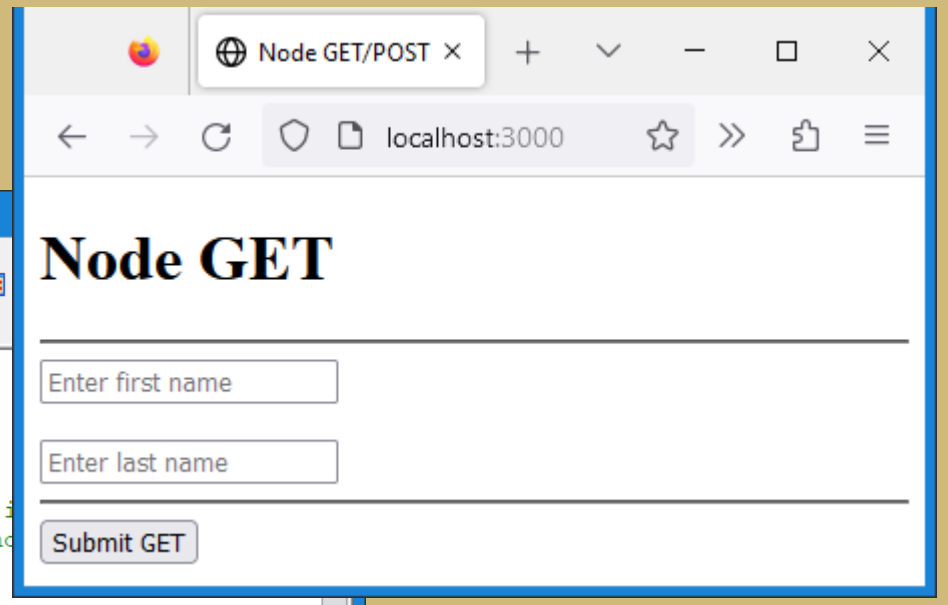
C:\Temp\CS386\Example21-3.js - Notepad++

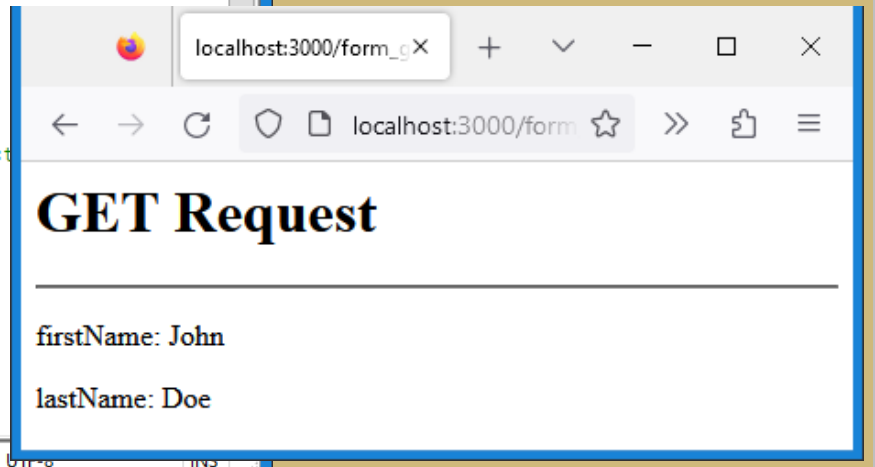
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example21-3.js

```javascript
const htmlForm = require('./modHtmlFormGET'); //Loading string html form
const http = require('http');//Load http module
const url = require('url');//Load url module
const port = 3000, host = 'localhost'; //Define host, port
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer i
    console.log(`Request method: ${req.method} and url: ${req.url}`);//Output request metho
    //Create variable objURL, parse request url with true argument for query string
    //Check for request method and url = /
        //Send html form
    //Check for request method GET and objURL pathname /form_get
        //Assign query property of objURL into constant objQS
        //Assign variable strHtml h1 with string 'GET Request'
        //Loop over query object objQS using prop as iteration variable
            //Accumulate iteration variable prop, colon and value of objQS wrapped in <p> into st
        //Ending for loop curly brace
        //Send strHtml
    }//Ending else if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,374 lines : 30 | Ln : 30 Col : 1 Pos : 1,375 | Windows (CR LF)





21.2 Request Object

➢ **Example 21-3:**

C:\Temp\CS386\Example21-3.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example21-3.js

```javascript
const htmlForm = require('./modHtmlFormGET'); //Loading string html form
const http = require('http');//Load http module
const url = require('url');//Load url module
const port = 3000, host = 'localhost'; //Define host, port
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into server constant
    console.log(`Request method: ${req.method} and url: ${req.url}`);//Output request method and url
    objURL = url.parse(req.url,true); //Create variable objURL, parse request url with true argument for query string
    if (req.method === 'GET' && req.url === "/"){ //Check for request method and url = /
        res.end(htmlForm); //Send html form
    } else if (req.method === 'GET' && objURL.pathname === "/form_get") { //Check for request method GET and objURL pathname /form_get
        const objQS = objURL.query; //Assign query property of objURL into constant objQS
        strHtml = "<h1>GET Request</h1><hr />"; //Assign variable strHtml h1 with string 'GET Request'
        for (let prop in objQS) { //Loop over query object objQS using prop as iteration variable
            strHtml += "<p>" + prop + ": " + objQS[prop] + "</p>"; //Accumulate iteration variable prop, colon and value of objQS wrapped in <p> into strHTML
        } //Ending for loop curly brace
        res.end(strHtml); //Send strHtml
    };//Ending else if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,713 lines : 30 | Ln : 30 Col : 1 Pos : 1,714 | Windows (CR LF) | UTF-8 | INS

21.2 Request Object

Request Body

- When receiving POST or PUT request, request body might be important for application
- Processing body data is bit more involved than accessing request headers
- Request object that is passed in to handler implements ReadableStream interface

- What are Streams?
  - Streams are one of fundamental concepts that power Node.js applications
  - Data-handling methods used to read or write input into output sequentially
  - Streams are way to handle data transports efficiently:
    - Reading/writing files
    - Network communications
    - Any kind of end-to-end information exchange
  - What makes streams unique:
    - Instead of program reading file into memory all at once (traditional way):
      - Streams read chunks of data piece by piece, processing its content without keeping it all in memory

21.2 Request Object

Request Body

- Get data right out of stream by listening to stream's 'data' and 'end' events
- Chunk emitted in each 'data' event is nodejs Buffer type
- For string data, convert each chunk into string and accumulate into variable
- Body assembled in this way looks like querystring

Syntax:

```
let body = ''; //Initialize variable body
req.on('data', function (chunk) {
    body += chunk.toString();
})
req.on('end', function() {
    res.end(body);
});
```

21.2 Request Object

- **Example 21-4:**
- Complete following steps in Example21-4.js:

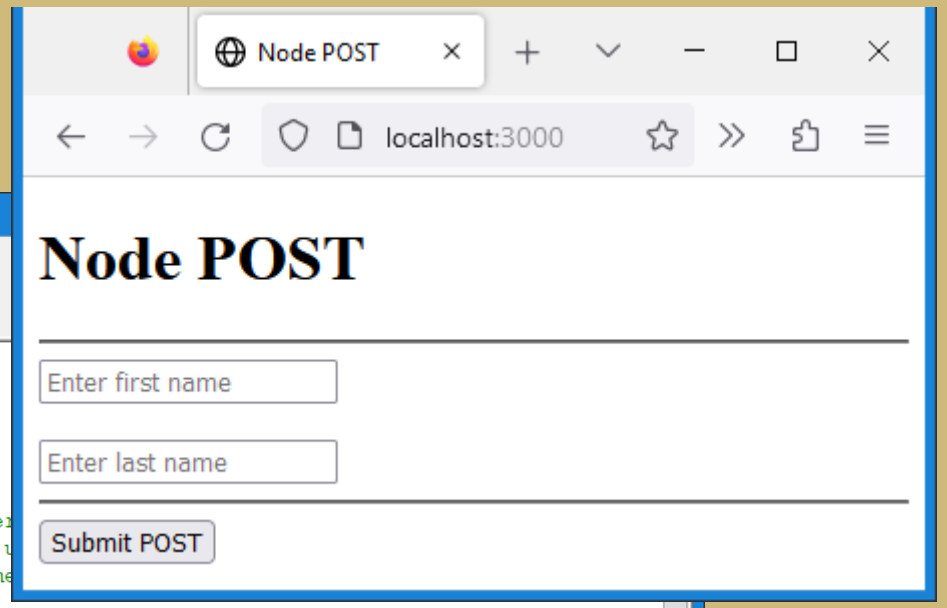
C:\Temp\CS386\Example21-4.js - Notepad++

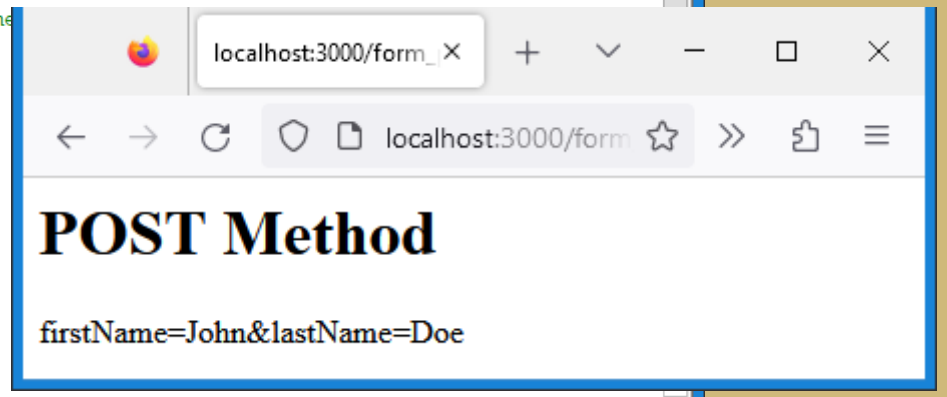
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example21-4.js

```
const htmlForm = require('./modHtmlFormPOST');
const http = require('http');//Load http module
const url = require('url');//Load url module
const port = 3000, host = 'localhost'; //Define host, port
//const parse = require('querystring').parse;//Use parse method from querystring module
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into ser
    console.log(`Request method: ${req.method} and url: ${req.url}`);//Output request method and u
    objURL = url.parse(req.url,true); //Create variable objURL, parse request url with true argume
    if (req.method === 'GET' && req.url === "/"){ //Check for request method and url = /
        res.end(htmlForm); //Send html form
    } else if (req.method === 'POST' && objURL.pathname === "/form_post") { //Check for request me
        //Initialize variable body to empty string
        //Listen to data event using on method, callback function with parameter chunk
            //Accumulate chunk converted to string into variable body
        //Ending function curly brace, on method parenthesis
        //Listen to end event using on method using callback function
            //Send h1 with string 'POST Method' and concatenate variable body
        //Ending function curly brace, on method parenthesis
    } //Ending else if curly brace
}) //Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,726 lines : 32 | Ln : 31 Col : 1 Pos : 1,725 | Windows (CR LF) | UTF-8 | INS





21.2 Request Object

➢ **Example 21-4:**

C:\Temp\CS386\Example21-4.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example21-4.js

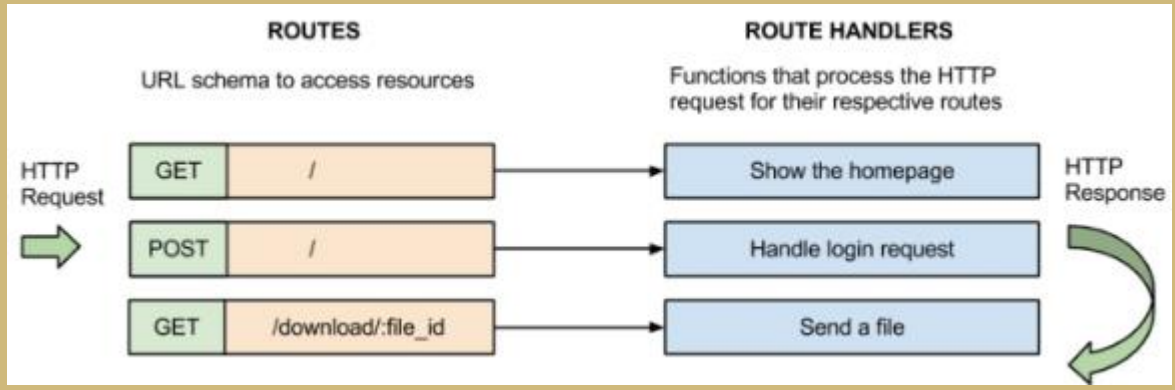
```javascript
const htmlForm = require('./modHtmlFormPOST');
const http = require('http');//Load http module
const url = require('url');//Load url module
const port = 3000, host = 'localhost'; //Define host, port
//const parse = require('querystring').parse;//Use parse method from querystring module
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into server constant
    console.log(`Request method: ${req.method} and url: ${req.url}`);//Output request method and url
    objURL = url.parse(req.url,true); //Create variable objURL, parse request url with true argument for query string
    if (req.method === 'GET' && req.url === "/"){ //Check for request method and url = /
        res.end(htmlForm); //Send html form
    } else if (req.method === 'POST' && objURL.pathname === "/form_post") { //Check for request method POST and and objURL pathname /form_post
        let body = ''; //Initialize variable body to empty string
        req.on('data', function (chunk) { //Listen to data event using on method, callback function with parameter chunk
            body += chunk.toString(); //Accumulate chunk converted to string into variable body
        }) //Ending function curly brace, on method parenthesis
        req.on('end', function() { //Listen to end event using on method using callback function
            res.end('<h1>POST Method</h1>' + body); //Send h1 with string 'POST Method' and concatenate variable body
        }) //Ending function curly brace, on method parenthesis
    } //Ending else if curly brace
}) //Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,874 lines : 32 | Ln : 32 Col : 1 Pos : 1,875 | Windows (CR LF) | UTF-8 | INS

21.2 Request Object

Routes

- Notice if statements in previous examples
- These are routes of web server
- What are routes?
  - Routes are URL schema, which describe interfaces for making requests to your web app
  - Combining HTTP request method (a.k.a. HTTP verb) and path pattern to define URLs in web app
  - Each route has associated route handler, which does job of performing any action in web app and sending HTTP response



21.2 Request Object

Routes

- Routes can also just serve static html pages
- Example:
  - "/" – index.html
  - "/about" – about.html
  - "/contact" – contact.html
  - "/public/stylesheet/style.css" – style.css
  - "/public/images/1.jpg" – 1.jpg
  - "/public/images/2.png" – 2.png
- Never map route to actual html or other resource file name
- Old way of implementing static web server, still in use today
- Reveals too many technical details:
  - Microsoft windows server
  - Active Server Pages (ASP) technology

21.2 Request Object

URL Encoding

- Characters in URL limited to defined set of reserved and unreserved US-ASCII characters
- Reserved characters:
  - Certain characters are reserved or restricted from use in URL because they may (or may not) be defined as delimiters by generic syntax in particular URL scheme
  - If reserved character is needed in another context (not URL) → must be encoded
  - Example:
    - Forward slash / characters are used to separate different parts of URL

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

- Unreserved characters:
  - Characters that are allowed in URL but do not have a reserved purpose are called unreserved
  - Unreserved characters include:
    - Uppercase and lowercase letters
    - Decimal digits
    - Hyphen, period, underscore, and tilde

21.2 Request Object

URL Encoding

- Any other characters besides reserved and unreserved are not allowed in URL
- But URL often contains characters outside US-ASCII character set → must be converted to valid US-ASCII format for worldwide interoperability
- URL-encoding (a.k.a percent-encoding) is process of encoding URL information so that it can be safely transmitted over internet
- To map wide range of characters that is used worldwide, two-step process is used:
  - At first data is encoded according to UTF-8 character encoding
  - Then only those bytes that do not correspond to characters in unreserved set should be percent-encoded like %HH, where HH is the hexadecimal value of byte
- Example:
  - François would be encoded as: Fran%C3%A7ois
  - https://www.w3schools.com/tags/ref_urlencode.ASP

21.2 Request Object

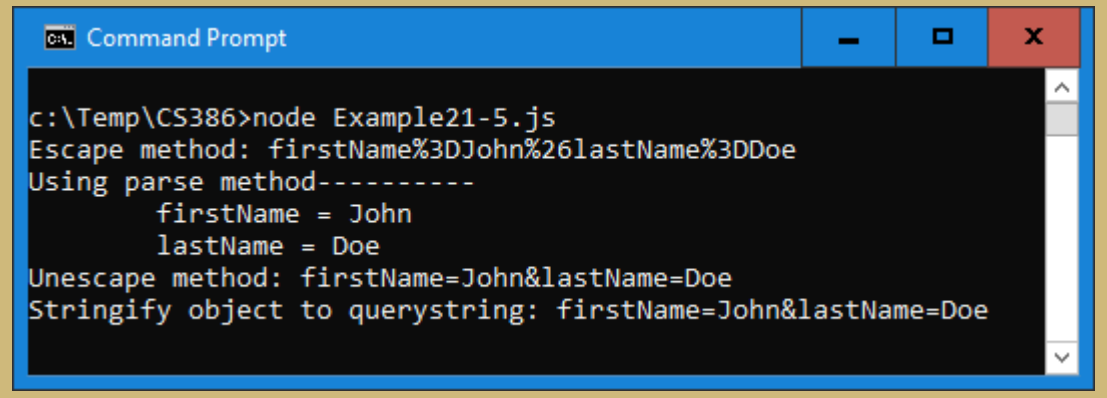
Core Module querystring

- Use core module querystring to perform operations on querystring type strings
- Based on previous example, parse entire querystring like request body into name/value pairs
- Also perform URL encoding (escape) and decoding (unescaped)
- Finally converts object into querystring format
- Querystring methods:

Method	Description
escape()	Returns an escaped querystring
parse()	Parses the querystring and returns an object
stringify()	Stringifies an object, and returns a query string
unescape()	Returns an unescaped query string

21.2 Request Object

- **Example 21-5:**
- Load querystring module into variable qs
- Create variable strQS to store:
  - "firstName=John&lastName=Doe"
- Escape strQS and display in console
- Create variable objQS and store return result from parse method using strQS
- Loop over object and display property and value
- Create variable strEscaped to store:
  - firstName%3DJohn%26lastName%3DDoe
- Use unescaped method using strEscaped and display in console
- Use stringify method using objQS and display in console



21.2 Request Object

➢ **Example 21-5:**

C:\Temp\CS386\Example21-5.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example21-5.js

```
1   const qs = require('querystring'); //Load querystring core module
2   const strQS = "firstName=John&lastName=Doe"; //Initialize querystring sample string
3
4   console.log('Escape method: ' + qs.escape(strQS)); //Escape method of qs on strQS in console.log
5   const objQS = qs.parse(strQS); //Create const objQS and assign parse method of qs on strQS
6
7   console.log('Using parse method----------');
8   for (prop in objQS) { //Loop over objQS using iteration variable prop
9       console.log('\t' + prop + " = " + objQS[prop]); //Display prop, = and value of objQS in console, tab indented
10  }
11  //Unescape method
12  //Escaped string, convert it into using unescape method
13  const strEscaped = 'firstName%3DJohn%26lastName%3DDoe'; //%3D --> = , %26 --> &
14  console.log('Unescape method: ' + qs.unescape(strEscaped));
15  //Stringify method
16  console.log('Stringify object to querystring: ' + qs.stringify(objQS));//Stringify object into querystring
```

JavaScript file | length : 930 lines : 20 | Ln : 20 Col : 1 Pos : 931 | Windows (CR LF) | UTF-8 | INS

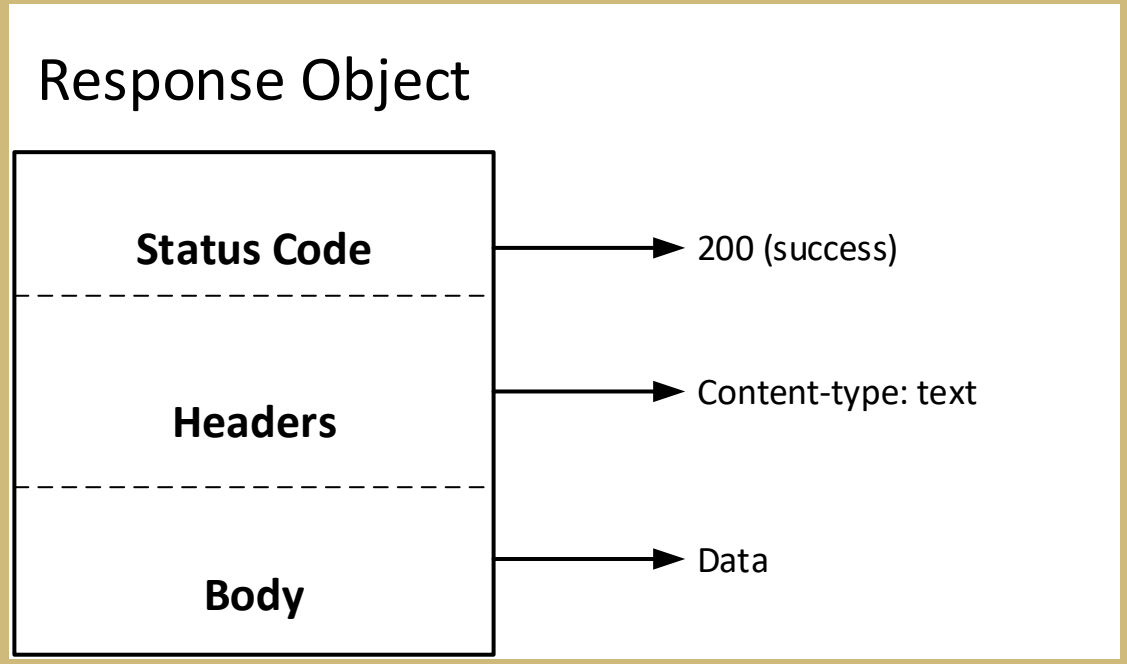
21.2 Request Object

- Properties and Methods (req is request object)

Properties/Methods	Description
req.accepts([types])	A convenience method to determine whether the client accepts a given type or types (optional types can be a single MIME type, such as application/json, a comma-delimited list, or an array).
req.body	An object containing POST parameters. It is so named because POST parameters are passed in the body of the REQUEST, not in the URL like querystring parameters. To make req.body available, you will need middleware that can parse the body content type.
req.cookies/ req.signedCookies	Objects containing containing cookie values passed from the client.
req.headers	The request headers received from the client.
req.hostname	A convenience method that returns the hostname reported by the client.
req.ip	The IP address of the client.
req.params	An array containing the named route parameters.
req.param(name)	Returns the named route parameter, or GET or POST parameters, avoid this method.
req.path	The request path (without protocol, host, port, or querystring).
req.protocol	The protocol used in making this request (for our purposes, it will either be http or https).
req.query	An object containing querystring parameters (sometimes called GET parameters) as name/value pairs.
req.route	Information about the currently matched route. Primarily useful for route debugging.
req.secure	A convenience property that returns true if the connection is secure. Equivalent to req.protocol=='https'.
req.url/req.originalUrl	These properties return the path and querystring (they do not include protocol, host, or port). req.url can be rewritten for internal routing purposes, but req.originalUrl is designed to remain the original request and querystring.
req.write(chunk[, encoding] [, callback])	To write string data into the body of the request object to send to the server.
req.xhr	A convenience property that returns true if the request originated from an AJAX call.

21.3 Response Object

- Response object (which is normally passed to callback)
- Can name it whatever you want:
  - Common to name it res, resp, or response
  - Starts its life as instance of http.ServerResponse (core Node object)
- Most common/important properties of response object



21.3 Response Object

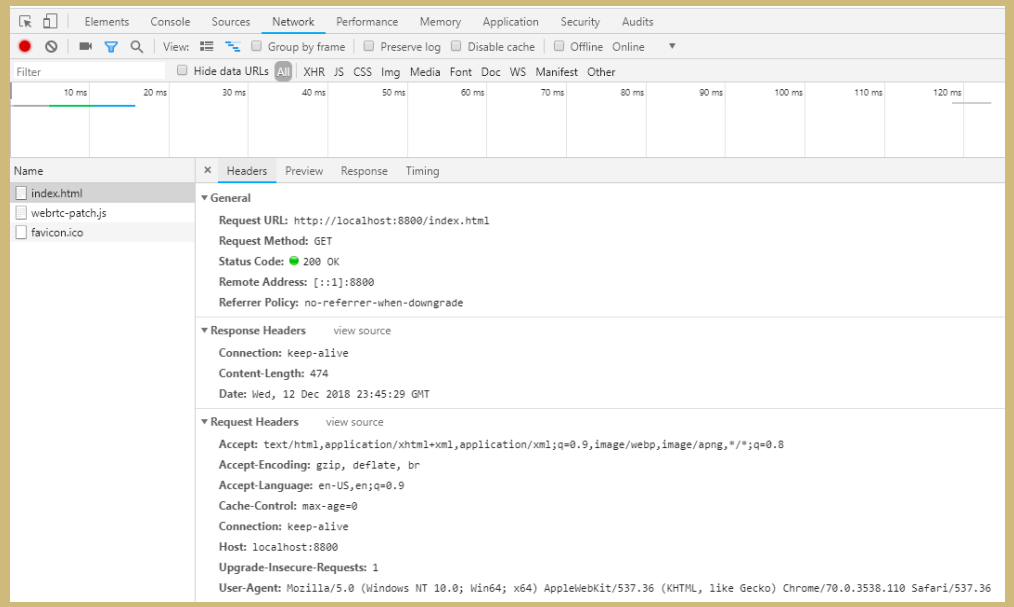
Response Headers

- Browser sends hidden information to server in form of request headers
- When server responds, it also sends information back
- Not necessarily rendered or displayed by browser
- Information typically included in response headers is metadata and server information
- Content-Type header:
  - Tells browser what kind of content is being transmitted (HTML, an image, CSS, JavaScript, etc.)
- In addition to Content-Type, there are other headers:
  - Whether response is compressed
  - What kind of encoding is used
  - Contain hints for browser about how long it can cache resource
- https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

21.3 Response Object

Response Headers

- Common for response headers to contain some information about server, indicating what type of server it is, and sometimes even details about operating system
- Downside about returning server information:
  - Gives hackers starting point to compromise your site
  - Extremely security conscious servers often omit this information, or even provide false information
- To see response headers: Browser's developer tools
- To see response headers in Chrome, for example:
  - Open JavaScript console (in general F12)
  - Click Network tab
  - Reload page
  - Pick HTML from the list of requests (it will be first one)
  - Click Headers tab → See all response headers



21.3 Response Object

- Properties and Methods (res is response object)

Properties/Methods	Description
res.end([data[, encoding]][, callback])	This method signals to the server that all of the response headers and body have been sent; that server should consider this message complete. The method, response.end(), MUST be called on each response.
res.getHeaders()	Returns a shallow copy of the current outgoing headers. Since a shallow copy is used, array values may be mutated without additional calls to various header-related http module methods.
res.status(code)	Sets the HTTP status code. Express defaults to 200 (OK), so you will have to use this method to return a status of 404 (Not Found) or 500 (Server Error), or any other status code you wish to use. For redirects (status codes 301, 302, 303, and 307), there is a method redirect, which is preferable.
res.set(name, value)	Sets a response header. This is not something you will normally be doing manually.
res.setHeader(name, value)	Sets a single header value for implicit headers. If this header already exists in the to-be-sent headers, its value will be replaced. Use an array of strings here to send multiple headers with the same name.
res.redirect([status], url)	Redirects the browser. The default redirect code is 302 (Found). In general, you should minimize redirection unless you are permanently moving a page, in which case you should use the code 301 (Moved Permanently).
res.writeHead(statusCode[, statusMessage][, headers])	Sends a response header to the request. The status code is a 3-digit HTTP status code, like 404. The last argument, headers, are the response headers. Optionally one can give a human-readable statusMessage as the second argument.

21.3 Response Object

- **Example 21-6:**
- Complete following steps in Example21-6.js:

C:\Temp\CS386\Example21-6.js - Notepad++

Example21-6.js

```javascript
const http = require('http');//Load http module
const port = 3000, host = 'localhost'; //Define host, port
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer int
    if (req.method === 'GET'){ //Check for request method
        //Set Content-Type (text/html) using setHeader method
        //Add header test property with value 'This is a test'
        //Store request headers object using getHeaders() method in constant resHeaders
        //Declare string variable strHeaders and initialize to empty string
        //Loop over all properties p in response header object resHeaders
            //Accumulate string of li elements using p and resHeaders[p]
        //Ending for loop curly brace
        //Use method writeHead to put http code 200 and string 'Example 21-6 Successful Request'
        //Send html string back using method end
        res.end(`<html<head><meta charset="UTF-8"></head><body><h1>Response Headers</h1>
        <p>Response header properties on ${new Date()}:</p>
        <ul> ${strHeaders} </ul>
        </body></html>`);//Merging strHeaders into html markup
    };//Ending if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,472 lines : 29 | Ln : 28 Col : 1 Pos : 1,471 | Windows (CR LF) | UTF-8 | INS

localhost:3000/

localhost:3000

Response Headers

Response header properties on Sat Mar 11 2023 06:27:04 GMT-0800 (Pacific Standard Time):

- content-type = text/html
- test = This is a test

21.3 Response Object

➢ **Example 21-6:**

C:\Temp\CS386\Example21-6.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

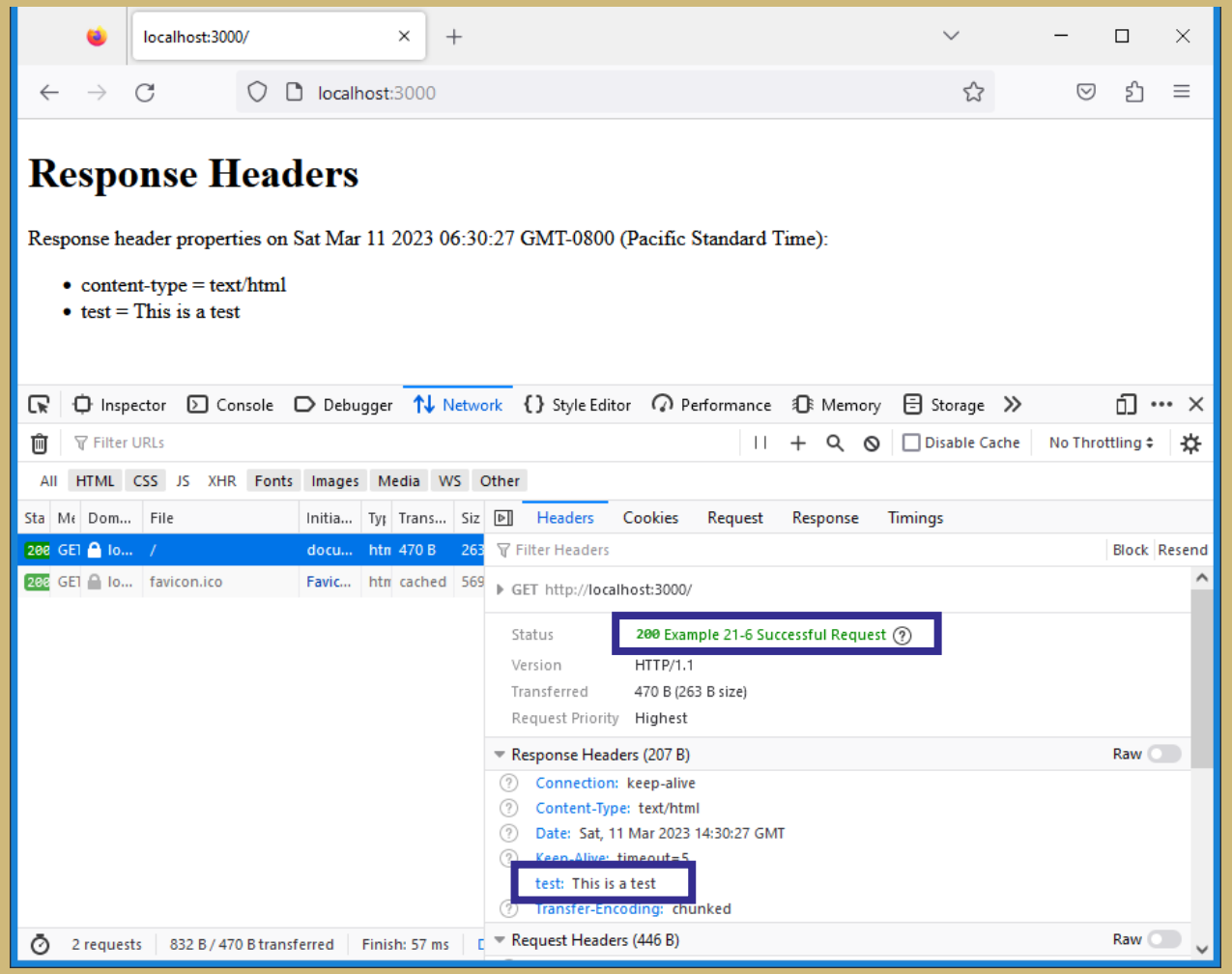
Example21-6.js

```javascript
const http = require('http');//Load http module
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into server constant
    if (req.method === 'GET'){ //Check for request method
        //Writing response headers
        res.setHeader('Content-Type', 'text/html'); //Set Content-Type using setHeader method
        res.setHeader('test', 'This is a test'); //Add header test property with value 'This is a test'
        const resHeaders = res.getHeaders(); //Store request headers object using getHeaders() method in constant resHeaders
        let strHeaders = ''; //Declare string variable strHeaders and initialize to empty string
        for (let p in resHeaders) { //Loop over all properties p in response header object resHeaders
            strHeaders += "<li>" + p + " = " + resHeaders[p] + "</li>"; //Accumulate string of li elements using p and resHeaders[p]
        } //Ending for loop curly brace
        res.writeHead(200, 'Example 21-6 Successful Request'); //Use method writeHead to put http code 200 and string 'Example 21-6 Successful Request'
        //Send html string back using method end
        res.end(`<html<head><meta charset="UTF-8"></head><body><h1>Response Headers</h1>
        <p>Response header properties on ${new Date()}:</p>
        <ul> ${strHeaders} </ul>
        </body></html>`);//Merging strHeaders into html markup
    };//Ending if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(3000, function() {//Starting server and listening for income connections
    console.log(`Server started on port 3000 on ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,702 lines : 27 | Ln : 27 Col : 1 Pos : 1,703 | Windows (CR LF) | UTF-8 | INS

21.3 Response Object

- **Example 21-6:**
- Open web developer tools
- Click on Network tab
- Reload page, then click on first item
- Notice the response headers



21.3 Response Object

- **Example 21-6 (advanced):**
- Add current date and time to new header property timestamp

C:\Temp\CS386\Example21-6_adv.js - Notepad++

Example21-6_adv.js

```javascript
const http = require('http');//Load http module
const port = 3000, host = 'localhost'; //Define host, port
//Create http server
const server = http.createServer(function(req, res){//Assign return value of createServer into server
    if (req.method === 'GET'){//Check for request method
        //Writing response headers
        res.setHeader('Content-Type', 'text/html');//Content-Type
        res.setHeader('test', 'This is a test');//test
        res.setHeader('timestamp', encodeURI(new Date().toLocaleString()) );//Need to encode date/time
        const resHeaders = res.getHeaders();//Store request headers object in variable
        let strHeaders = '';//Declare string variable
        for (let p in resHeaders) {//Loop over all properties p in response header object
            if (p === 'timestamp') { //Need to decode date/time value
                strHeaders += "<li>" + p + " = " + decodeURI(resHeaders[p]) + "</li>";
            } else {
                strHeaders += "<li>" + p + " = " + resHeaders[p] + "</li>";//Build string of li elemen
            }
        }//Ending for loop curly brace
        res.writeHead(200, 'Example 21-6 Successful Request');
        res.end(`<html<head><meta charset="UTF-8"></head><body><h1>Response Headers</h1>
        <p>Response header properties on ${new Date()}:</p>
        <ul> ${strHeaders} </ul>
        ${new Date().toLocaleString()}</body></html>`);//Merging strHeaders into html markup
    };//Ending if curly brace
});//Ending callback function curly brace, method parenthesis
server.listen(port, host, function() {//Starting server and listening for income connections
    console.log(`Server started:
    ${host}:${port}
    on: ${new Date}`);//Display console message when server started
});//Ending callback function curly brace, method parenthesis
```

JavaScript file | length : 1,757 lines : 34 | Ln : 33 Col : 1 Pos : 1,756 | Windows (CR LF) | UTF-8 | INS

localhost:3000/

localhost:3000

Response Headers

Response header properties on Sat Mar 11 2023 06:33:30 GMT-0800 (Pacific Standard Time):

- content-type = text/html
- test = This is a test
- timestamp = 3/11/2023, 6:33:30 AM

3/11/2023, 6:33:30 AM