Web Servers

In this lab, you will use most of the key components of the node.js http module.

# Objectives

In this lab, you will learn to

* create HTTP servers and make client requests,
* read and write data to and from the HTTP server.

# Start your webserver

1. Open webserver.js in your IDE. This essentially the same "Hello World" server as we saw in the introduction. Run this file from your IDE or from the command line. Navigate to <http://localhost:3000> with your browser and you should see "Hello World" in your browser.

# Add support for different clients

2. Shut down the server for now. We're going to add the ability to respond differently to different clients using the value of the "Accept" header. Add code to the requestListener() method to send "Hello World" back to the client in an H1 HTML element if the client accepts 'text/html.' You have access to all the headers on the request.headers property. Restart the server and you should now see an HTML version of "Hello World" in your browser.

You can also try to run curl http://localhost:3000, to see the example of plain response.

# Use parameters in the query string

## Add a supporting module

3. Shut down the server again. Include the url module by adding a requires statement and assigning it to a variable.

## Read the values in the query string

4. Go back to the requestListener() method and use the url.parse() method to parse the URL that is available on the request.url property. Use true for the second argument to the url.parse() method so that the query string will be available. Assign the return to a variable.

## Change the HTML tag using a query parameter

5. Change your code where you are writing out the HTML H1 element to use the value of a query parameter named style. If there is no value for the style parameter, use H1 as the default value.

## Test the query parameter

6. Navigate to <http://localhost:3000> to see "Hello World" still as an H1 element. Now add your query parameter to change it to a P, for example as in <http://localhost:3000>?style=p.

# Making a client request from node.js

Node.js also has nice support for initiating client requests to an HTTP server.

## Run the client

7. Open the file client.js in your IDE. This is essentially the same code as we saw in the lecture on client requests. With your web server code still running, run this file. You should see 'Hello World' print out in the console. There are no HTML tags since you added the branching based on the "Accept" header.

## Change the request method for the client

8. To make a POST request to the server, change the method option from GET to POST.

## Change the content type

9. Go to the request object and use request.setHeader() to set the value of the "Content-Type" header to "application/json."

## Write a JSON object to the request

10. After the header is set, use request.write() to write a JSON object to the request. The object should be an array of strings using the first names of the students around you.

# Handling POST requests on the server

## Add branching based on the request method in the request listener

11. Go back to the webserver.js file. In the createServer() function, replace the requestListener function with an inline function that will branch based on the value of the request.method property. If the value is 'GET,' call a function named handleGet(); if the value is 'POST,' call a function named handlePost(). Both functions should have the request and response as parameters.

## Create a function for handling GET requests

12. You've already implemented everything for handling GET requests in the requestListener() function. Just rename it to handleGet().

## Create a function for handling POST requests

13. Create a function name handlePost() that takes the request and response as parameters. Add the code to read the data that is in the request using the IncomingMessage interface and build a string object with it. Attach a listener for the 'data' and 'end' events.

## Write the response using the array that was posted to the server

14. When the 'end' event is fired, signaling that you have read all of the data from request, convert the JSON string into an array. Iterate over the contents of the array and build up a string of "Hello [value]" followed by a newline for every element of the array. Then write the string out to the response with a content type of "text/plain."

## Test the POST implementation

15. Restart your web server code and then go to the client request and run it with your new POST implementation. You should see "Hello [name]" printed on a new line for every name in the array.