**Table of Contents**

[Table of Contents](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week6/practical6-step1.html#toc0)

[Creating a Simple NodeJS Express Server](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week6/practical6-step1.html#toc1)

[Step 1 - Create the GitHub Repository](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week6/practical6-step1.html#toc2)

[Step 2 - Create the Express Server](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week6/practical6-step1.html#toc3)

[Step 3 - Upload the Code and Test](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week6/practical6-step1.html#toc4)

**Creating a Simple NodeJS Express Server**

In this practical you will learn how to create an HTTP server - which you will use for the rest of the module to serve data.

\*\* NB – The work done in these practicals will be directly used for your assignment – make sure to follow the instructions correctly \*\*

Don't forget to commit and push your code very regularly.  You will need to use git add -A to add any new files to GitHub

So far, you have been using the app.js code that was already in the GitHub repositories to serve the HTML and Javascript code you have written.    This is a simple *express* server whose job it is to serve files from any directory.  In this server, we use

app.use(express.static(\_\_dirname));

to serve the files requested.

However, we also need another application server, to handle data requests – i.e. provide access to data stored in our database.  We’ll store this code in a totally separate repository as in practice a data API can be used by many different applications.  This data API will provide the functionality to interact with the database – e.g. save the data and also generate GeoJSON based on the latest version of data stored in the database.  You will create this application server yourself in this practical.

**Step 1 - Create the GitHub Repository**

1.       If you have not already done so, accept the second assignment code – see separate sheet for where to find this.

2.       Create a branch called *express*

3.       Clone the repository onto your Linux server – don’t forget to use –b to clone the new branch

**Step 2 - Create the Express Server**

1.       Create a file called dataAPI.js

2.   Add the following code into the file - the *require* command allows NodeJS to reference external code

"use strict";

/\*\* Express router providing user related routes

\* @module dataAPI

\* @requires express

\* @requires path

\*

\*/

/\*\*

\* express module

\* @description express is the server that forms part of the nodejs program

\* @const

\*/

const express = require('express');

/\*\*

\* path module

\* @description path is the module that logs the requests made to the router

\* @const

\*/

const path = require("path");

/\*\*

\* Express router to mount user related functions on.

\* @type {object}

\* @const

\* @namespace dataAPI

\*/

const dataAPI = express();

3.  Then add some code to actually create the server.   Note that we are assigning the server a fixed PORT number - this is mapped back to the https://<<your server name>>/api link that is set up in Apache on the server.

// add an http server to serve files

let http = require('http');

let httpServer = http.createServer(dataAPI);

let port = 4480;

let server = httpServer.listen(port);

4.   We then need to add some code to allow CORS - this is Cross Origin Resource Sharing.   What that means is that we can server data from https://<<your server name>>/api  and combine it with data or HTML or code from another website (in our case https://<<your server name>>/api

// adding functionality to allow cross-domain queries

dataAPI.use(function(req, res, next) {

res.setHeader("Access-Control-Allow-Origin", "\*");

res.setHeader("Access-Control-Allow-Headers", "X-Requested-With");

res.setHeader('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');

next();

});

5.  Now add the code to send a basic response when the user types in https://<<your server name>>/api

/\*\*

\* Route serving basic test message

\* @name /

\* @function

\* @memberof module:dataAPI

\* @inner

\*/

dataAPI.get('/',function (req,res) {

res.send("hello world from the Data API on port: "+port);

});

6. Add some code to log all the requests to console.log so that you can track what is happening on the server

// adding functionality to log the requests

dataAPI.use(function (req, res, next) {

let filename = path.basename(req.url);

let extension = path.extname(filename);

console.log("The file " + filename + " was requested.");

next();

});

**Step 3 - Upload the Code and Test**

1. Upload the server and add it to GitHub and commit and push your changes.  The dataAPI.js file should go in the following directory - i.e. the root directory of your GitHub API code.

/home/<<your CS username>>/code/<<your github API repository name>>

2. Run the server in debug mode (so that you can see any errors on the console) by typing

cd /home/<<your CS username>>/code/<<your github API repository name>>

node dataAPI.js

Hint:  If you are DEBUGGING/WORKING ON your API code, always use the NODE command to run the API.  If the code is debugged and working fine, you can use PM2.

3.       Test the server by typing https://<<your server IP address>>/api  into a browser

You should see ‘hello world from the Data API’  (or a similar message)

4.       Stop the server by typing:

ctrl-c