**Table of Contents**

[Table of Contents](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step1.html#toc0)

[Basic PostGIS Connectivity](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step1.html#toc1)

[Step 1 - Install the PG Module Using NPM](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step1.html#toc2)

[Step 2 - Create a Route for the PostGIS Functions](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step1.html#toc3)

**Basic PostGIS Connectivity**

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

NB - It is very important that you understand API and Route and End Point theory before you start work on these practicals!

In this practical we will build a simple API route and end point to query data from the PostGIS database and return it as text to the user.     We will start this off by generating a list of Points of Interest.

**Step 1 - Install the PG Module Using NPM**

As with any other programming language, JavaScript allows us to take advantage of code that others have written.  In client-side JavaScript (in the browser) we use *<script>* tags.   For server-side JavaScript - NodeJS - we use the *requires* key word to reference the code.

1. Keep working in the branch from the previous week

2. At command line, go to the root directory of your API code and type the following to install the PostgreSQL / PostGIS package that contains all the code we need to connect to the database.

npm install pg --save

The save option means that package.json will be updated so that when you next clone the repository and type npm install the pg package will be installed as well.

3. Don't forget to commit and push

**Step 2 - Create a Route for the PostGIS Functions**

While it is possible to continue writing code in the dataAPI.js file we used in the previous practical, it is best practice to put code in separate files as it makes things easier to debug.    We will use a ROUTE approach (see lecture slides) to easily separate out different functionality.

1. Create a file called geoJSON.js

2. Add the following code at the top of the file - this code calls all the external Node modules we will need for our route

"use strict";

let express = require('express');

let geoJSON = require('express').Router(); // create a new route

let pg = require('pg'); // the code to connect to PostgreSQL

let fs = require('fs'); // code to read the database connection details file

3.  Add a test end point - this doesn't actually do anything except return the data that you type into the URL, but it is very useful way of testing whether your code is working  (similar to an alert message for client side JavaScript).

geoJSON.route('/testGeoJSON').get(function (req,res) {

res.json({message:req.originalUrl});

});

4.  At the very bottom of the file, add the following code.   The module.expors function in NodeJS is a good way to implement the *encapsulation* concept from Object-Oriented programming.  If another user references your code (using *require* they will only be able to reference the code listed on the exports list.  In our case, we are exporting geoJSON, which is the express router - so that means that all the end points for that router are also available)

module.exports = geoJSON;

5. Upload the geoJSON.js file to a sub-directory called *routes* - i.e. to the following structure

/home/<<your CS username>>/code/<<your GitHub API repository name >>/routes

5.  We now need to link from the main express server to this one, so that if this new route is passed to the web server as a request it will be directed correctly.  At the bottom of dataAPI.js, add the following

NB: you will need to CHANGE the route to the one listed on Moodle for this module.

const geoJSON = require('./routes/geoJSON');

dataAPI.use('/geoJSON', geoJSON);

6.   Test your code - you should use the following URL, adapted to your new route name

https://<< your CS computer name >>/api/<<new route name>>/testGeoJSON?name=claire&surname=ellul

7. The response should be a message with the URL parameters that you put after the question mark