**Table of Contents**

[Table of Contents](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc0)

[Basic PostGIS Connectivity](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc1)

[Step 1 - Create a File with Your Database Connection Details](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc2)

[Step 2 - Create the PostgreSQL/PostGIS Pool Connection](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc4)

[3. Create a Route for a Basic PostGreSQL/PostGIS Query](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc5)

[Appendix 1 – Testing Your PostgreSQL/PostGIS Connection  - Checklist](http://cege0052.cege.ucl.ac.uk/cege0043tutorials/tutorials/week7/practical7-step2.html#toc6)

**Basic PostGIS Connectivity**

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

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 - Create a File with Your Database Connection Details**

We don't want to store any database connection details in our code, as the code is stored in GitHub and can be seen by GitHub administrators.  So we need to store our database usernames and passwords in a separate file.

NB:  This file  MUST NOT BE added to GitHub - that would be a major security breach!

1. Create a file called *postGISConnection.js*

2. Add the following content to the file, in exactly the order shown - make sure to use your own username and password, and make sure you include the commas as shown

host: cege0052.cege.ucl.ac.uk,

user: userXXX,

database: ucfscde,

password: cege0052XXXXXX,

port: 5432

3. Save the file

4. Upload it to your server and add it to the following directory

/home/<< your CS username>>/certs

**Step 2 - Create the PostgreSQL/PostGIS Pool Connection**

Note that the code here is not in a function - that means it will be run when the file is loaded, i.e. as soon as we start the API server.

A pool connection is a permanent connection to the database that can then be re-uesd multiple times in your code - i.e. every time you need to run some SQL.

We will use the database connection file created above, and process each line to get the connection parameters we need.

1. Add the following code to the geoJSON.js file, just above the *testGeoJSON* end point. This code will retrieve your username from the Linux operating system, so that the code can work out where the connection details file is stored

// we need to get the user's name from the login used

// to connect to the server

// that way we can work out the correct path for the connection file

// /code/<<username>>/certs/

const os = require('os');

const username = os.userInfo().username; // locate the database login details

console.log(username);

2.  Read the file using the fs (file system) package  - the path to the file includes the user's login name from the previous step

// oad the database login details file

let configtext = ""+fs.readFileSync("/home/"+username+"/certs/postGISConnection.js");// locate the database login details

3. Loop through the file line by line, extract the login information, create a configuration and start the database pool

// now convert the configruation file into the correct format -i.e. a name/value pair array

// this means looping through the file looking for commas

// each comma indicates a new line, a new piece of information

// we then take the information and convert it into a configuration

// for the PostgreSQL connection

let configarray = configtext.split(",");

let config = {};

for (let i = 0; i < configarray.length; i++) {

let split = configarray[i].split(':'); //split = split one text string into two

config[split[0].trim()] = split[1].trim(); //trim = remove any spaces before or after the text

}

let pool = new pg.Pool(config);

console.log(config);

**3. Create a Route for a Basic PostGreSQL/PostGIS Query**

We will use the information\_schema table in our query.  This gives us some details about the tables in the database (you may remember that a database is "self-describing", this is what is meant by that statement - you can query the database to find out what is in the database)

1. Add the following code underneath the previous testGeoJSON  end point.  This will connect to the database and - if the connection works - run the SQL and return the result  to the browser.

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

// create a new connection in the pool

// the connection will return err if it failes

// if it works, the connection will return a client called client - which can be

// used to run some SQL

// done is the name of the function to be called once the SQL has

// returned a value - this closes the connection so that it can be

// reused

pool.connect(function(err,client,done) {

if(err){

console.log("not able to get connection "+ err);

res.status(400).send(err);

}

// the SQL that we want to run

let query = "select \* from information\_schema.columns";

// pass the SQL to the client from the pool

// will return err if it fails

// result will hold any values returned by the SQL

client.query(query ,function(err,result) {

done();

if(err){

console.log(err);

res.status(400).send(err);

}

// send to send the result back to the browser

// result.rows will give an array of all the rows

// in the result

res.status(200).send(result.rows);

});

});

});

**Note** the following (see class notes for a full list of status values)

**res.status(200)** - everything has worked well

**res.status(400)** - there has been an error

2. You can test your code using the following URL

https://<<your CS machine name>>/api/<<route name>>/postgistest

You will see a long list of the tables and schemas in the database.

**Appendix 1 – Testing Your PostgreSQL/PostGIS Connection  - Checklist**

Here's a check list for if your database connection doesn't work ..

•       Is the connection data correct in your postGISConnection.js file?  E.g. is the IP address of the hostname correct, are you connecting to the ucfscde database?

•       Is that file located in the  correct location

•       Is the file parsing correctly when you start the dataAPI.js server?

•       Can you create a connection pool or do you get an error message at the console?

•       Can you run the SQL you generate?  Test this by using console.log to echo out the SQL and copy/paste into PGAdmin