# Individual Programming Project - Part 2

**Due: 23:59 Friday 12 May 2017**

**Submission: Please submit a readme file, your curl test commands, and a zip of your project directory (source for the website + service) through the online submission portal (https://apps.ecs.vuw.ac.nz/submit/NWEN304)**

**This project is worth 15%**

## Overview

You are required to extend Part 1 of this project by creating a REST service that provides your website with content. Using Node.js and Express, two industry standards, you will design and implement a web service with a RESTful interface. You will need to modify your Part 1 solution to use Ajax calls to communicate with the service. The service should be supported by a postgresql database.

This project will take place in four parts:

1. Create and run a Node.js service locally to serve your website content.
2. Connect your website with your server via appropriate Ajax calls.
3. Construct a Postgresql database to maintain the state of the service.
4. Host the service on the cloud service, Heroku.

## Marking

The project will be marked out of 100. The following describes the distribution of marks.

**Core (65%)**

- (20%) Define an appropriate REST interface for your server
- (20%) Implement a Node.js service with the REST interface
- (20%) Implement Ajax calls to send and receive data from your REST service
- (5%) Define a set of test cases (using the curl command) that demonstrate the correct functioning of your REST service.

**Completion (25%)**

- (10%) Create a local postgresql database to support your service
- (10%) Use the database to enable persistent information storage in your REST service.
- (5%) Implement appropriate error catching for database transaction/query handling.

**Challenge (10%)**

- (10%) Deploy the service successfully to Heroku so your REST service can be accessed anywhere in the Internet.

## Background

**Node.js** is an open source runtime environment for developing web applications. It is optimised for scalability and throughput. Node.js is used extensively by industry leaders, including Netflix and Linkedin.

More information and documentation can be found at: https://nodejs.org/en/

**Express.js** is a Node.js framework for building web applications. It is the defacto standard for building Node.js applications and provides utility methods for accelerating their creation. In this project, you will use Express.js to build a RESTful web service.

More information and documentation can be found at: http://expressjs.com/

**Postgresql** is an object-oriented relational database management system. It is primarily used as a database server to store data securely and enables retrieval of data through SQL. It is designed to handle workloads ranging from a single user to enterprise level web applications. Using SQL you can create, insert, delete, and query data in a postgresql database.

**Heroku** is Platform as a Service (PaaS) cloud provider. It allows you to host web applications developed in various programming languages. You can use Heroku as a git repository to upload your content. It also provides the capability of hosting a postgresql database.

## Core – create a Node.js service

Design and implement a simple REST service on your local machine. The lab machines have Node.js installed on them. If you are running this from home, you will need to install Node.js before running your service.

### 1.1 Building the service

Create a working directory for the Node.js service and use npm init to perform the initial setup. It will ask you for configuration settings, either set these or press enter to use the default options.

```
> npm init
```

Now install Express.js in the directory (this may already be done by the lab machines):

```
> npm install express --save
```

### 1.2 Create an index.js file for your service

Enter the following:

```
var express = require('express');
var app = express();
var port = process.env.PORT || 8080;

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(port, function () {
  console.log('Example app listening on port 8080!');
});
```

### 1.3 Run the service with node

```
> node index.js
```

That should bring up your service with the console log saying it is listening on port 8080.

### 1.4 Testing and debugging your service

There are a few ways that you can test to see if your service is working as intended. Web based may be easiest to start with, but as you start posting data to your service it becomes more complicated to accurately and reliably test. Curl is a command line tool that you can use to generate HTTP requests and pass data to your service.

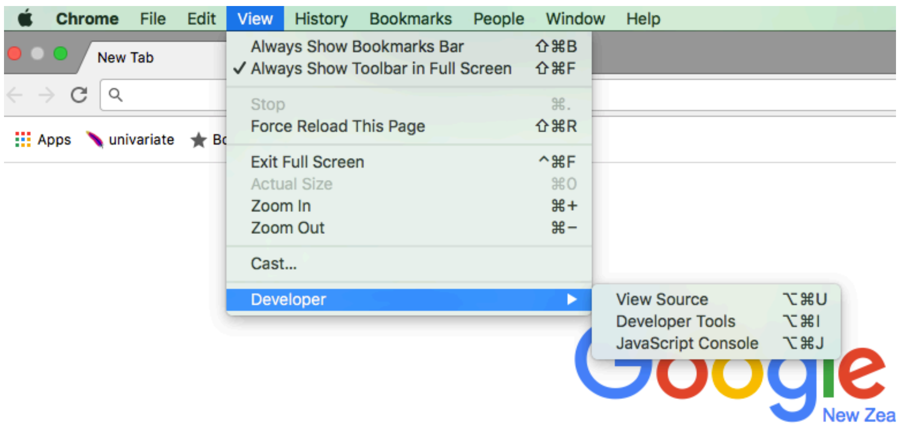
Web based:
http://localhost:8080/

Curl:
```
> curl localhost:8080
```

Curl can also be used to perform other HTTP operations, such as POST and PUT. E.g.,

```
> curl -H "Content-Type: application/json" -X POST -d '{"username":"xyz","password":"xyz"}'
http://localhost:3000/api/login
```

To debug errors you can consider using the console.log() function. You may also consider using your browser's development tools. You can bring up the development tools in Google Chrome as shown below.



### 1.5 Define an appropriate REST interface

This should conform to a RESTful architecture, making use of appropriate resource identifiers (i.e. URLs) and suitable operations (i.e. HTTP request methods).

### 1.6 Define test cases

Define a set of curl-based test cases to demonstrate that your service is working correctly. There should be at least one curl command for each of your API function according to your REST interface design. These must be submitted and will be used for marking.

### 1.7 Create Ajax calls to utilise your REST service

Implement appropriate Ajax calls to send and receive data between your web page and your REST service.

Here is an example of posting JSON data with Ajax:

```
var ERROR_LOG = console.error.bind(console);
$.Ajax({
        method: 'POST',
        url: 'http://localhost:8080/my_post_function/',
        data: JSON.stringify({
            task: task.find('.task').html()
        }),
        contentType: "application/json",
        dataType: "json"
    }).then(my_next_function, ERROR_LOG);
```

**NOTE:**
You may experience a problem with cross-site scripting, where you will not be allowed to invoke functions on a service hosted in a different location (e.g. if you do not rely on your Node.js server to serve the web page that makes the above Ajax call). One way to get around this problem is to add the following to your REST service.

```
// Add headers
app.use(function (req, res, next) {
  // Website you wish to allow to connect
  res.setHeader('Access-Control-Allow-Origin', '*')

  // // Request methods you wish to allow
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH,
DELETE');

  // Request headers you wish to allow ,
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Access-Control-Allow-
Headers');

  // Pass to next layer of middleware
  next();
});
```

## Completion – support the service with a PostgreSQL database

RESTful services should be stateless, meaning it should not have user data stored in variables etc. Instead, it should rely on a database to maintain the state of the service. You are required to build a small postgresql database and have the service read and write data from it.

There will be a lecture on postgresql database in Week 7. Your aim is therefore to complete the core part before week 7.

Instructions on how to set up a local postgresql database on the lab machines can be found on the projects page.

However, a quick summary is:

1. On a lab machine you need to type ‘need postgres’. The databases are stored on a machine called Depot (not your local host). This is important to know when connecting your service to the database.
2. You can then create a database with ‘createdb <username>_nodejs
3. Then you can connect to it with the ‘psql <dbname>’, e.g. ‘psql <username>_nodejs’.
4. You should change your database password (to something other than your ecs account password) with: ‘alter user <username> with password 'XyZZy';’
5. To exit the postgres interpreter type ‘\q’.
6. To list tables in your database use \dt or \d <table name>.

**2.1 Create a simple database**

Your database does not need to be very complex. Connect to your database and create a single table with an id and a task. This can be extended later if necessary.

```
> create table todo (id serial primary key, item varchar(255));
```

Insert a value into the table:

```
> insert into todo (item) values ('first todo item');
```

Query the table to check it is now in there:

```
> select * from todo;
```

You will later need to modify your table to store whether a job has been completed or not. This is done with an SQL update command.

**2.2 Connect to your database in your service**

```
var pg = require('pg');
var connectionString = "postgres://myuser:mypassword@depot:5432/mydatabase";

var client = new pg.Client(connectionString);
client.connect();
```

Then make a function to query the data and return the result:

```
app.get('/test_database', function(request, response) {
   // SQL Query > Select Data
   var query = client.query("SELECT * FROM todo");
   var results = []
   // Stream results back one row at a time
   query.on('row', function(row) {
      results.push(row);
   });

   // After all data is returned, close connection and return results
   query.on('end', function() {
      client.end();
      response.json(results);
   });
});
```

**2.3 Use your database in the service**

Implement appropriate database calls for each API function of your REST service.

**2.4 Implement appropriate error checking**

Ensure both your web page and service are robust by performing error checking and handling throughout them. You should consider invalid database transactions, invalid Ajax requests, invalid Ajax responses, etc.

## Challenge – host your TODO service in the cloud

**3.1 Host your service on Heroku**

There will be a lecture on Heroku in Week 7/8. Your aim is therefore to finish the completion part in week 7.

Sign up for Heroku and use the free tier to host your service. You will need to read the Heroku documentation (linked below) to work out how to do this.

https://www.heroku.com/

## Things for submission

You should submit the following through the online portal (https://apps.ecs.vuw.ac.nz/submit/NWEN304):

1. A readme file explaining:
    a. How to use your system.
    b. The design of your REST interface with good justifications.
    c. What error handling has been conducted and why.
2. A zip of your entire project directory, including the source for the web page and REST service.
3. The test cases that you defined in Section 1.6.