Exercise 1

Create a simple JSON HTTP server

Prior Knowledge
Unix Command Line Shell
Some simple JavaScript (node.js)

Learning Objectives
Understand the basics of a Web Server

Software Requirements
Node.js
Npm
A Text Editor (e.g. Atom)

Creating a node.js program

1. Node.js is an effective framework for writing server-side programs using the JavaScript language. In this exercise we are going to create a simple program that returns a random number between 1 and 100.

   Because we expect the result to be read by a machine not a human, we will return this as a JSON not as an HTML.

2. Make a directory called ex1. You can do this by starting a terminal window and typing:
   ```
   mkdir ~/ex1
   cd ~/ex1
   ```

3. Now we need to create a new node.js project:
   `npm init`

   This creates a simple JSON file that the Node Package Manager uses to keep track of your project. We need to fill in some details (but actually we can just accept the defaults. Just hit **Enter** for all the entries.

```
oxsoa@oxsoa:~/ex1$ npm init
This utility will walk you through creating a package.json
file.
It only covers the most common items, and tries to guess
sensible defaults.

See `npm help json` for definitive documentation on these
fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package
and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (ex1)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/oxsoa/ex1/package.json:

{
  "name": "ex1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)
oxsoa@oxsoa:~/ex1
```

4. This uses a library called express.js which is a very popular framework for writing REST applications in JS. Let's install that:
   `npm install express -save`

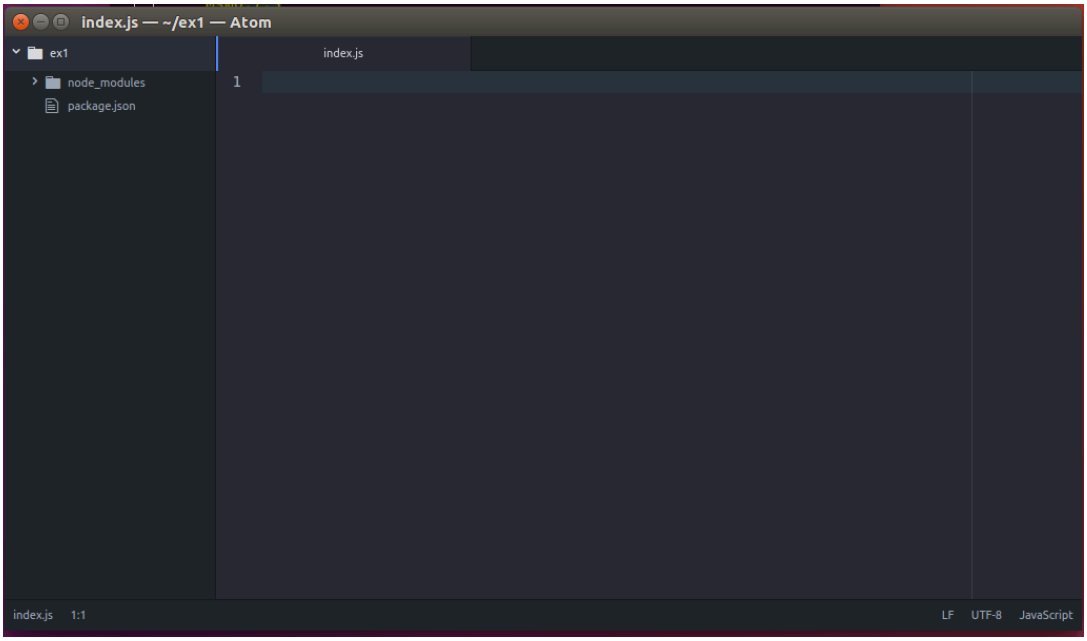
   You will see a lot of colourful text scroll by, ending in a couple of warnings:
   `npm WARN ex1@1.0.0 No description`
   `npm WARN ex1@1.0.0 No repository field.`

   You can ignore these.

5. Now we need to create a file and code the server.
   In the terminal window type:
   ```
   atom index.js
   ```

   Hint: If you have another text editor on Ubuntu that you prefer, switch to that instead.

6. You should see an Atom editor window. You may need to close various welcome windows:



7. Type (or copy and paste) the following code.

   The code is at http://freo.me/ex1-js
   If you copy and past please make sure you understand the code.

```
var http = require('http'),
    express = require('express'),
    app = express();

app.get("/",function(req,res){
    obj = {random : Math.floor((Math.random() * 100) + 1)};
      res.json(obj);
});

var server = app.listen(8080, function() {
    console.log("Random server listening on port 8080");
});
```

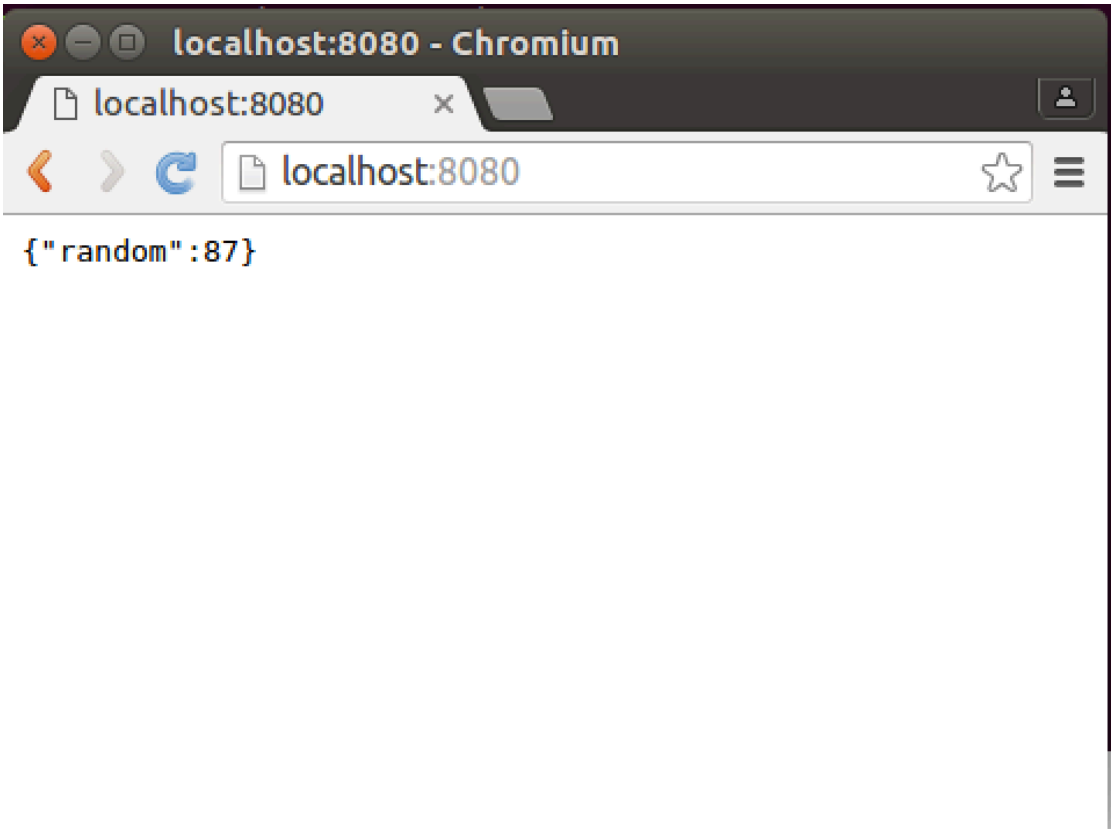
8. This code creates an HTTP server that responds to any HTTP GET request in the same way. It will instantiate a JavaScript object containing a random number and then return that as a JSON string.

9. To run this code, you need to type the following into a terminal window:
```
cd ~/ex1
node index.js
```

   You should see the server respond:
```
oxsoa@oxsoa:~/ex1$ nodejs index.js
Random Server listening on: http://localhost:8080
```

10. You can test this code by pulling up a browser window (e.g. Chromium or Firefox) and then browsing to http://localhost:8080

    The result is this:



11. However, we do not want a human-/browser-enabled service. We want to call this service from machine-based clients. Let's first try curl (a command-line URL / HTTP tool).

    Start a new terminal window (hint Right-click on the icon) and then type:
    ```
    curl http://localhost:8080
    ```

    You should see:
    ```
    curl http://localhost:8080
    {"random":71}oxsoa@oxsoa:~/ex1$
    ```

    Hint: Because the HTTP response has no '\n' line ending, the result is a bit hard to read as the next line merges with the output.

12. curl provides a useful debug facility. If you turn on verbose output, you can see the actual network messages as they are sent on the wire:

    curl –v http://localhost:8080
    You should see output similar to this:

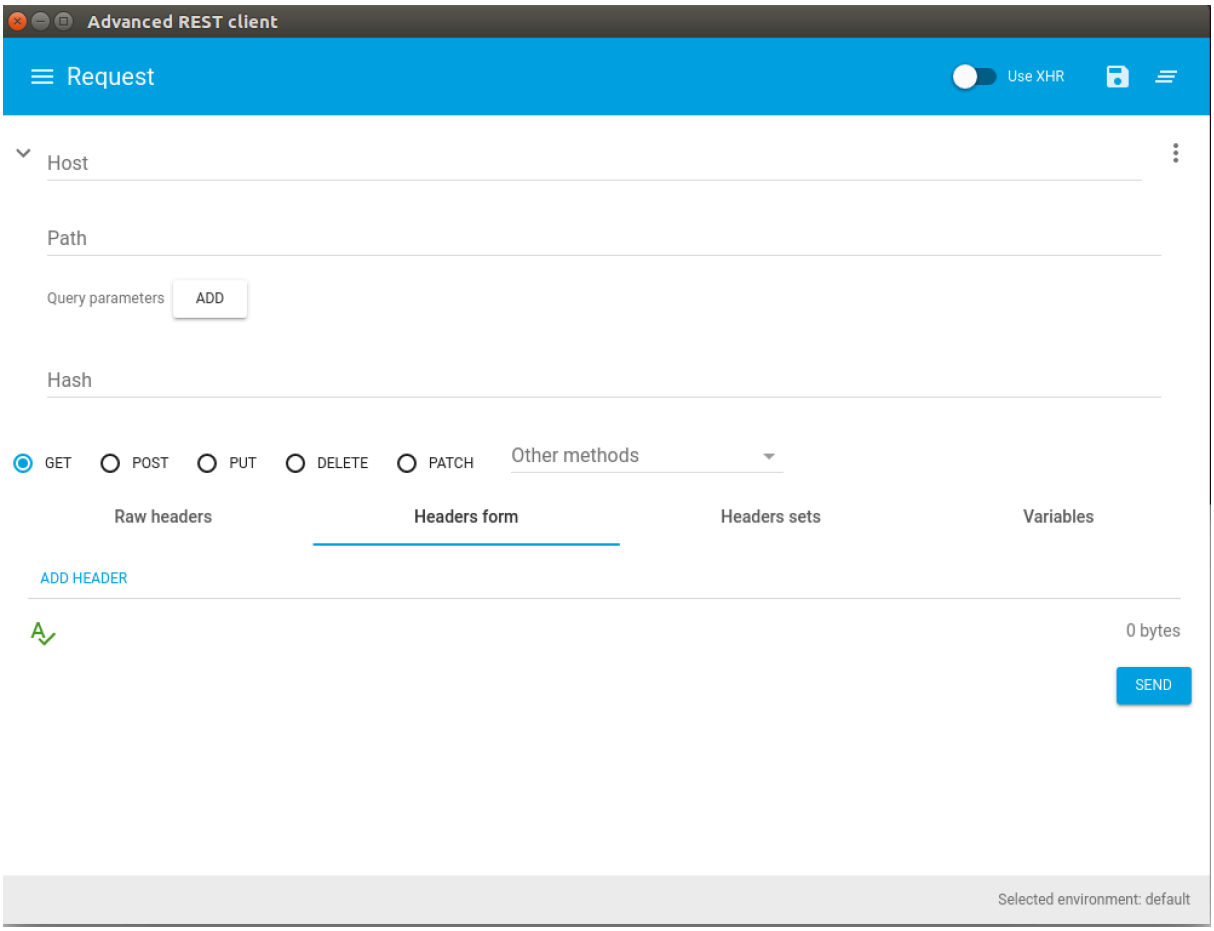
    ```
    * Rebuilt URL to: http://localhost:8080/
    *   Trying 127.0.0.1...
    * Connected to localhost (127.0.0.1) port 8080 (#0)
    > GET / HTTP/1.1
    > Host: localhost:8080
    > User-Agent: curl/7.47.0
    > Accept: */*
    >
    < HTTP/1.1 200 OK
    < Content-Type: application/json
    < Date: Tue, 24 May 2016 09:04:03 GMT
    < Connection: keep-alive
    < Content-Length: 13
    <
    * Connection #0 to host localhost left intact
    {"random":33}oxsoa@oxsoa:~/ex1$
    ```

    The lines beginning with > indicate that these are sent to the server and < are received from the service.

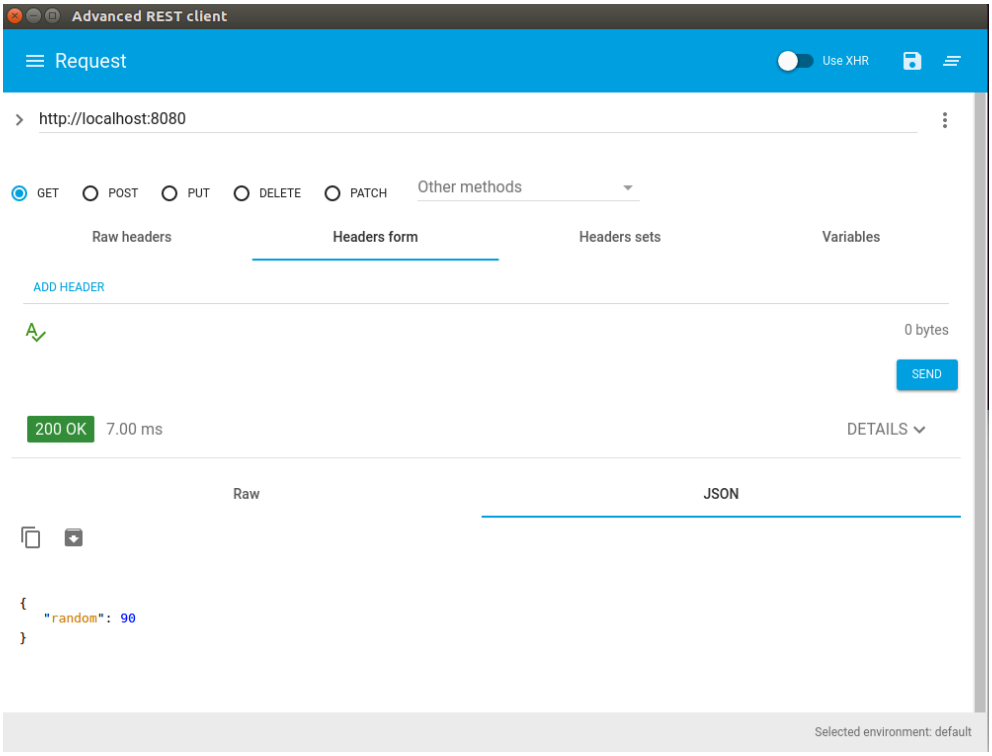
13. Another way of testing this is to use a tool called the Advanced Rest Client (ARC) in Chrome/Chromium. Start Chromium and open up a new window or tab. In the corner is a little button called Apps 

Click on that and then choose the ARC button:

14. You should see a window like this. There may be some old history in there.



15. Type http://localhost:8080 into the Request URL field. Choose GET and then click Send. You should see:



16. *Automated testing of the service*

    We want this service to meet a set of behavior requirements. To ensure this, we can use a set of tests. There are a number of testing frameworks for SOA services. For this example, we are going to use a JavaScript tool called Frisby (http://frisbyjs.com/), which builds on top of another node.js test framework called Jasmine.

    I have written a test script for this service. It is available as a gist on Github. You can download it onto your VM using the following command:

```
cd ~/ex1
curl -L http://freo.me/ex1-test -o simplehttp_spec.js
```

17.

The test script looks like this:
The test does an HTTP GET on the URL and then validates the following aspects:

```
var frisby = require('frisby');

frisby.create('Test Random Number service')
  .get('http://localhost:8080/')
  .expectStatus(201)
  .expectHeaderContains('Content-Type',
'application/json')
  .expectJSONTypes( {
    random: Number
    }
  )
  .expectJSON({
    random: function(v) {
expect(v).toBeGreaterThan(0);expect(v).toBeLessThan(101);
}
  })
.toss();
```

a. The return code is 201
b. The Content-Type header is “application/json”
c. The JSON type of the result is a number
d. The JSON contains a tag called random, with a value >0 and <101

18. You can run this test using:
```
jasmine-node .
```

19. *Does the result match your expectations?*

20. Let's fix the server so that it passes the test, or the test to match the server. I'll leave this up to you.

    Hint: you will need to stop and restart the node server once you have edited the code.

21. Once the tests are passing, this exercise is complete.

    Recap:
    We have created a simple http server that returns a JSON output. We have tested this service in a number of ways – including via browser, ARC, curl and through a proper automated test.

    In our next exercise we will create a client for this service.