PRELUDE - SECURITY

Before we being, you might like to see if someone is trying to break into your server!

```
$ cat /var/log/auth.log | grep Failed
```

We should probably do something about that. A simple safe guard is to slow down brute force attacks by banning their IP for an amount of time. The default installation of 'fail2ban[1]' will do this for sshd port 22.

```
sudo apt-get update
sudo apt-get install fail2ban
```

PART 3 (...continued from last week) – HTTP VERBS

During the previous workshop we made a ReSTlike server using Python's Tornado library and sqlite3 as storage. You might remember that when we finished I said that even though the server “worked”, in terms of our ReSTlike goals the design was terrible. Why was that? (hint: read Richardson's Maturity Model – Level 2 – HTTP Verbs. http://martinfowler.com/articles/richardsonMaturityModel.html)

All our HTTP requests are currently GETs, and GETs should be idempotent, a “safe operation” (meaning in this instance that they should not change the contents of the database). There are other HTTP verbs...

Note. The usual CRUD operations (Create, Read, Update, Delete)
    are *not* the same as the HTTP verbs POST, GET, PUT, DELETE

“The comparison of the database oriented CRUD operations to HTTP methods has some flaws. Strictly speaking, both PUT and POST can create resources; the key difference is that POST leaves it for the server to decide at what URI to make the new resource available, whilst PUT dictates what URI to use; URIs are of course a concept that doesn't really line up with CRUD. The significant point about PUT is that it will replace whatever resource the URI was previously referring to with a brand new version, hence the PUT method being listed for Update as well. PUT is a 'replace' operation, which one could argue is not 'update'.” - `http://en.wikipedia.org/wiki/Create,_read,_update_and_delete`

Although a RESTlike design is good enough for our purposes, do feel free to read up on this...

```
http://www.restapitutorial.com/lessons/httpmethods.html
https://stormpath.com/blog/put-or-post/
```

1 https://en.wikipedia.org/wiki/Fail2ban

==================
ATTEMPT - 2
==================

Here is a better design, the VERB (GET/PUT) should come from the http request
Most of the code is the same, so copy 5.py to 6.py and edit as below...

```
import tornado.httpserver
import tornado.ioloop
import tornado.web
import sqlite3

_db    = sqlite3.connect('mytest.db')
_cursor = _db.cursor()

class dataRequestHandler(tornado.web.RequestHandler):
    def get(self):
        _cursor.execute("SELECT * FROM myData")
        for row in _cursor:
            self.write(str(row[0])+"<br>")

    def put(self):
        newValue = [int(self.get_argument("value"))]
        _cursor.execute("INSERT INTO myData VALUES (?)",newValue)
        _db.commit()
        self.write('OK')

application = tornado.web.Application([
    (r"/data", dataRequestHandler),
])

if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(43210)
    tornado.ioloop.IOLoop.instance().start()
```

TEST IT - You will now need a browser plugin to test
e.g. `https://addons.mozilla.org/en-US/firefox/addon/httprequester/`

```
GET http://178.62.91.44:43210/data
PUT http://178.62.91.44:43210/data?value=2
```

NOTE - This is better, but is still not RESTful in many different ways... (e.g. HATEOAS)
However, a full (level 3) implementation of a RESTful design is beyond the scope of this module.

Our current server only has a single resource that we can refer to ('data'). Next we will look at having multiple resources and how different data can be associated with each.

PART 4 - FINAL TORNADO SERVER EXAMPLE

For our simple API we are going to assume we have multiple sensors, and that we want to store the data from these along with a timestamp. The follow examples show what we would like our API to look like:
1) put a new timestamped value for a specific object, e.g. for a sensor with id 5:

```
PUT http://myhost/sensor/5?value=10&time=1
```

2) get values for a specific sensor between two timestamps
e.g.

```
GET http://myhost/sensor/5?time=0,100
```

3) delete all data (initialise the database)
e.g.

```
DELETE http://myhost/sensors/all
```

===================
Much of the code is the same as before, so copy 6.py to 7.py and edit as below...

0) First, let's start a new database file (since the structure is now substantially different)

```
-_db = sqlite3.connect('mytest.db')
+_db = sqlite3.connect('sensors.db')
```

```
-class dataRequestHandler(tornado.web.RequestHandler):
```

...and remove the rest of this class (we'll start a new request handler class).

```
+class sensorRequestHandler(tornado.web.RequestHandler):
```

...methods for DELETE, PUT and GET to follow.

1) DELETE (initialise) the database

```
-    (r"/data", dataRequestHandler),
+    (r"/sensors/all", sensorRequestHandler),
```

```
+    def delete(self):
+        _cursor.execute("DROP TABLE IF EXISTS data")
+        _cursor.execute("CREATE TABLE data (ID INT, value REAL, time INT)")
+        _db.commit()
+        self.write('OK')
```

TEST IT

```
DELETE http://178.62.91.44:43210/sensors/all
```

Should return OK and the sensors.db file should be created.

2) PUT new data

```
+    (r"/sensor/([0-9]+)", sensorRequestHandler),
```

```
+    def put(self, ID):
+        record = (int(ID), float(self.get_argument("value")), int(self.get_argument("time")))
+        _cursor.execute("INSERT INTO data VALUES (?,?,?)",record)
+        _db.commit()
+        self.write('OK')
```

TEST IT

```
PUT http://178.62.91.44:43210/sensor/1?value=22&time=10
PUT http://178.62.91.44:43210/sensor/1?value=99&time=20
PUT http://178.62.91.44:43210/sensor/1?value=10&time=30
```

Should all return OK and the data appear in sensors.db.
You can check sensors.db using sqlite3, e.g.

```
$ sqlite3 sensors.db
sqlite> .table
data
sqlite> select * from data;
…
```

3) GET data

```
+import sys

+    def get(self, ID):
+        range = self.get_argument("range",default="0,"+str(sys.maxint)).split(',')
+        params = [ID]+range
+        _cursor.execute("SELECT * FROM data WHERE ID=? AND time>=? AND time<=?", params)
+        records = []
+        for row in _cursor:
+            records = records + [{'ID':row[0],'value':row[1],'time':row[2]}]
+        self.write(tornado.escape.json_encode(records))
```

TEST IT

```
GET http://178.62.91.44:43210/sensor/1
```

Should return all the data for sensor #1 (formatted in JSON)

```
GET http://178.62.91.44:43210/sensor/1?range=15,25
```

Should return a time limited set.

Summary (for comparison final minimal version of our RESTlike code)

```
import tornado.httpserver
import tornado.ioloop
import tornado.web
import sqlite3
import sys

_db = sqlite3.connect('sensors.db')
_cursor = _db.cursor()

class sensorRequestHandler(tornado.web.RequestHandler):
    def delete(self):
        _cursor.execute("DROP TABLE IF EXISTS data")
        _cursor.execute("CREATE TABLE data (ID INT, value REAL, time INT)")
        _db.commit()
        self.write('OK')
    def put(self, ID):
        record = (int(ID), float(self.get_argument("value")), int(self.get_argument("time")))
        _cursor.execute("INSERT INTO data VALUES (?,?,?)",record)
        _db.commit()
        self.write('OK')
    def get(self, ID):
        range = self.get_argument("range",default="0,"+str(sys.maxint)).split(',')
        params = [ID]+range
       _cursor.execute("SELECT * FROM data WHERE ID=? AND time>=? AND time<=?", params)
       records = []
        for row in _cursor:
            records = records + [{'ID':row[0],'value':row[1],'time':row[2]}]
        self.write(tornado.escape.json_encode(records))

application = tornado.web.Application([
    (r"/sensors/all", sensorRequestHandler),
    (r"/sensor/([0-9]+)", sensorRequestHandler),
])

if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(43210)
    tornado.ioloop.IOLoop.instance().start()
```

PART 5 - DRAWING A GRAPH

The following simplified example is included to show how your service could be used to provide data for a web page (in this instance, a basic rendering of a Google Line Chart).

==================

The server just requires on extra line, so copy 7.py to 8.py and edit as below...

```
+    (r'/static/(.*)', tornado.web.StaticFileHandler, {'path': 'static'}),
```

This allows files put into the 'static' directory to be served directly by filename.
Make the directory and make it the current working directory.

```
$ mkdir static
$ cd static
```

We then need some client side javascript to load the data into a google chart. You should be able to download this from my example server (i.e. save it in your `static` folder as `graph1.html`, and you will of course need to adjust the url to point to your own data server).

```
<html>
  <head>
    <!--Load the AJAX API-->
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script type="text/javascript">

      // Load the Visualization API and the piechart package.
      google.load('visualization', '1.0', {'packages':['corechart']});

      // Set a callback to run when the Google Visualization API is loaded.
      google.setOnLoadCallback(loadData);

      // Callback to load the json and instantiate the data table
      function loadData() {
         jQuery.getJSON( "http://178.62.91.44:43210/sensor/1", function(records) {

            var data = new google.visualization.DataTable();
            data.addColumn('datetime', 'Time');
            data.addColumn('number', 'Value');

            for (index = 0; index < records.length; ++index) {
               record = records[index];
               var record_date = new Date(parseInt(record['time'])*1000);
               data.addRow([record_date, record['value']]);
            }
            drawChart(data);
         });
      }

      // Callback that instantiated the chart and draws the data table
      function drawChart(data) {

      var options = {
        width: 1000,
        height: 563,
        hAxis: {
          title: 'Time'
        },
        vAxis: {
          title: 'Value'
        }
      };

      var chart = new google.visualization.LineChart(
        document.getElementById('chart_div'));

      chart.draw(data, options);
      }
    </script>
  </head>

  <body>
    <center>
    <h1>Sensor 1</h1>
    </center>
    <!--Div that will hold the pie chart-->
    <div id="chart_div"></div>
  </body>
</html>
```

TEST IT, e.g. point your browser at your equivalent to:

`http://178.62.91.44:43210/static/graph1.html`