

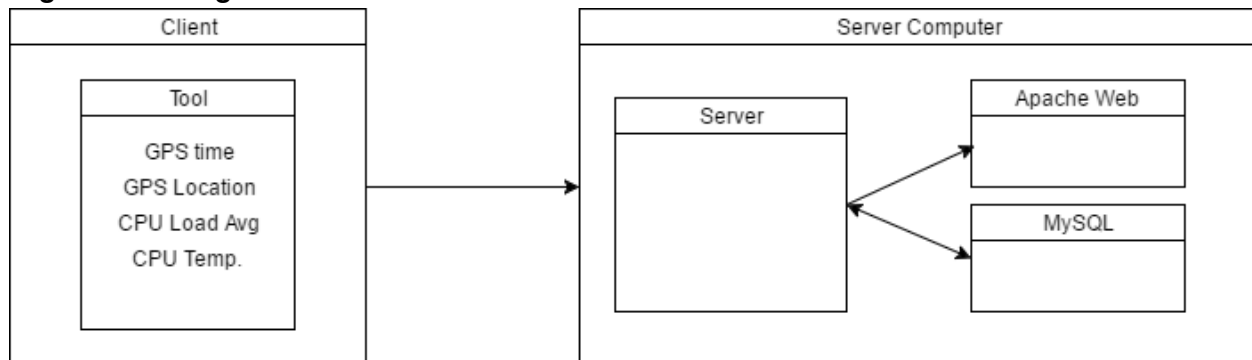
Project Summary:

This project will use JavaScript, Apache Web Server, Html, Canvasjs, MySQL, and php to make a client/server set of tools that work together. The client will scan a computer and report cpu load averages and timestamps. This data is then sent to a node.js server using either MQTT. The server will then take that data and store it into an sql database and is supposed to generate graphs using the Canvasjs library. An Apache Web Server will host a website that will act as a front end displaying the generated graphs in the user's web browser.

This assignment is based on a similar project from another class, but the original project was much less in-depth. Our aim is make something grander, but given time limits of the project and pre-existing knowledge of our group we feel that the above description is a good tradeoff between learning new skills and demonstrating existing ones.

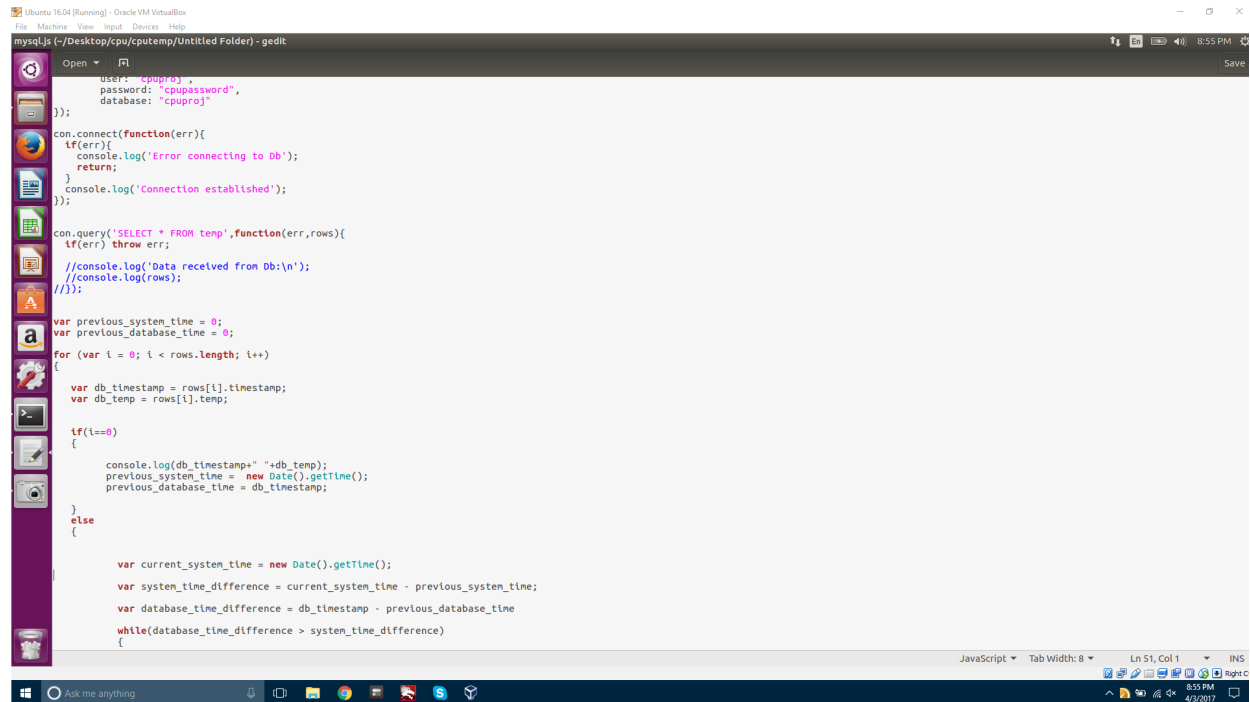
Motivations:

Our primary motivation for this project is to complete all requirements. In order to accomplish this, we originally decided to use an Raspberry Pi for our server, in order to get some experience with using the device. But do to time constraints and conflicts, we decided against this and decided to host our server on Ubuntu running a laptop.

High level design:

A client accesses the server with the tool protocol which contains the system info such as CPU load, temp, ect. and connects to the server. The server then takes the data from the client, stores it into a SQL database, and makes a chart with the statics from the clients.

Low level design:



```

mysqljs (-/Desktop/cpu/putemp/Untitled Folder) - gedit
user: 'cpuproj',
password: 'cpupassword',
database: 'cpuproj'
});
con.connect(function(err){
  if(err){
    console.log('Error connecting to Db');
    return;
  }
  console.log('Connection established');
});
con.query('SELECT * FROM temp',function(err,rows){
  if(err) throw err;
  //console.log('Data received from Db:\n');
  //console.log(rows);
  //});
var previous_system_time = 0;
var previous_database_time = 0;
for (var i = 0; i < rows.length; i++)
{
  var db_timestamp = rows[i].timestamp;
  var db_temp = rows[i].temp;

  if(i==0)
  {
    console.log(db_timestamp+" "+db_temp);
    previous_system_time = new Date().getTime();
    previous_database_time = db_timestamp;
  }
  else
  {
    var current_system_time = new Date().getTime();
    var system_time_difference = current_system_time - previous_system_time;
    var database_time_difference = db_timestamp - previous_database_time
    while(database_time_difference > system_time_difference)
  }
}

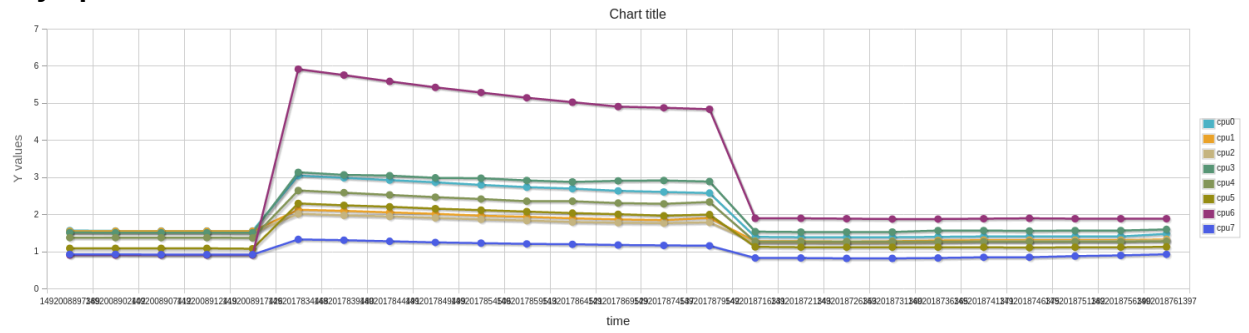
```

The client/"tool" sends cpu load averages to server using MQTT. The server receives data from the client and stores the data into a MySQL database. The Apache Web Server will host a website that will act as the front end and display the graphs being generated by the server. MySQL will store the data and PHPmyadmin is a user interface for the MySQL database.

Validation:

For the cpu load averages the only way to validate right now is to compare the server console log and the sql database. Both the Server and the Client are validated and the MySQL database is storing and querying as it should. Unfortunately we have not been able to get our php code working with our web page so we have not been able to successfully generate a graph. We hope to have the graph in phase 2. That being said we did generate a line graph in phpmyadmin to demonstrate it.

Mysql Demo



Server and Client Demo

```

scn@scn-GE72-2QD: ~/glt/CPSC4240Project
warningCount: 0,
message: '',
protocol41: true,
changedRows: 0 }
{
  hostname: 'scn-GE72-2QD',
  cpu0: '1.67',
  cpu1: '4.34',
  cpu2: '4.3',
  cpu3: '4.24',
  cpu4: '3.25',
  cpu5: '2.77',
  cpu6: '2.9',
  cpu7: '2.69',
  time: '1492226796622' }
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 2,
  warningCount: 0,
  message: '',
  protocol41: true,
  changedRows: 0 }

run time = 0
CPU 5:
  user 2.32
  nice 0
  sys 0.44
  idle 97.23
  irq 0
CPU 6:
  user 2.41
  nice 0.03
  sys 0.47
  idle 97.1
  irq 0
CPU 7:
  user 2.23
  nice 0.01
  sys 0.45
  idle 97.31
  irq 0
{"hostname": "scn-GE72-2QD", "cpu0": "1.67", "cpu1": "4.34", "cpu2": "4.3",
 "cpu3": "4.24", "cpu4": "3.25", "cpu5": "2.77", "cpu6": "2.9", "cpu7":
 "2.69", "time": "1492226796622"}

if int(i[2]) == 1:
    start = i[1]

```

Security:

Mysql: We hardened the mysql server using `mysql_secure_installation`. This includes setting a strong randomly generated password for root, removing anonymous users, Disallowing root login remotely, removing test databases, and installing the `validate_password` plug-in which allows you to change password requirements. Then we set them to high.

Client and Server: The biggest thing here is to have the client encrypt data and then have the server decrypt it and store it on the local mysql database. However given the nature of our client and server we can encrypt to prevent from man in the middle attacks, but our code is exposed since it is javascript which means there isn't much we can do to hide our encryption methods to our users. I would like to do this using ssl/tls but it will be more practical to use the `crypto` module and make our own. MQTT doesn't have encryption built in so the data needs to be encrypted before it leaves the client and decrypted on arrival at the server.

Server Expressjs Webserver: We Installed the `helmet` module which provides protection against well known web vulnerabilities by setting the HTTP headers appropriately. Helmet does this by installing nine smaller middleware functions that set security-related HTTP headers. This includes `csp`, `hidePoweredBy`, `hpkp`, `hsts`, `ieNoOpen`, `noCache`, `noSniff`, `frameguard`, `xssFilter`.

Dependencies: Dependencies give nodejs a lot of power however they can also be a source of insecurity so I installed two vulnerability scanners `nsp` check and `snyk`. `Nsp` check will only check for vulnerabilities and give suggestions. `Snyk` will scan for vulnerabilities and then patch or update for you.