SYSTEM RESOURCE MONITOR (SRM) PROJECT SCOPE

September 26, 2016

OVERVIEW

1. Project Background and Description

There are a lot of PC running in a network. There will be different persons using the PC for different purpose. The PC resource utilization varies up on time. For the system admin there is a need to monitor system usage and make decision based on that. The same way PC user has also be intimated where there is excess of resource usage from their side. Excessive usage happens due to running high performance application or due to hardware error or malware application. The best way to detect Hardware or software issues is to monitor the system resource usage.

When there is large number of PC in a network the resource usage data should be used for future analysis. So the data has to be stored in a centralized server.

2. Project Scope

Instead of each user looking at the task manager and seeing if there is any issue, an app can be used to the job. It works on the background and gathers memory, CPU and process usage. These three parameters defines the resource usage of a PC. Since the resource usage data is needed for the system admin and for later analysis it has to be a client server architecture. When there is excess resource usage in a system the system user has to be intimated that there excess usage. For that alert mails are used. When a PC has excess resource usage that allowed by the system admin the user of the pc receives an alert notification mail. This helps the PC user to be intimated that there may be some issue with the PC. The PC resource utilization limits will wary from PC to PC and by time. So the limits are to be configurable at the server side.

There has to be a server where the system admin can specify limit parameters. Based on that server has to decides whether any pc crossed the limits. When the limit is crossed the end user has to be intimated by a mail. The configurable limit parameters should be placed in an xml. Whenever the xml parameters are updated the system has to make the decision based on the updated value. The client parameters should be logged to a database for later analysis. The detailed analysis of the data on database is beyond the scope of the project.

The general scope of client and server side is as follows

i) Client App
   (1) The boundary of the application is to run in LAN environment where there will be 100s of PC.
   (2) The machines are to be monitored for basic system level parameters like RAM, CPU usage and number of process running at an instance.
   (3) Each PC should have a unique identification needed.
   (4) The resource should be monitored regularly at least in every 5 minutes. Whenever the client sends data to the server, based on the configuration data in the server the decisions are made.
   (5) The client should log the messages to a local log file.
   (6) The analysis of the logged data is not in the scope of the project.

ii) Server App
   (1) Installed on a single machine on the same intranet. This can act at the server.
   (2) The client send information to the server and the server process the message.

(3) The received message from e ach client has to check for validity. If it is valid it is logged to a RDMS server.
(4) The server app monitors the data for abnormality.
(5) Abnormality are mentioned in xml file.
(6) Each client parameters are compared with values mentioned in the xml files. When an abnormality is observed and alert should be triggered.
(7) When an alert is triggered, it is displayed in the console and an alert mail is send.
(8) There has to have an alert mail template. The template should give the user whatever information needed to make him understand what happened.
(9) Alert template has to be a generic html template used for all alerts.
(10) Whenever an alert mail is send the configuration parameters for that PC and the values send from the client area also mentioned in tabular format for better understanding.
(11) The analysis of the logged data in the RDMS server is not in the scope of the project.

3. High-Level Requirement Analysis

The new system must include the following:

Client App:

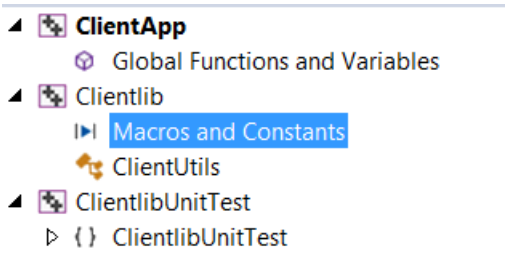
- Ability to run 100's of copies of the software in a LAN.
- Ability to specify server IP when the application starts executing.
- Ability to uniquely identify each client PC. Each machines hostname is the unique identifier.
- Ability to get RAM usage in real-time.
- Ability to get CPU usage at real-time.
- Ability to get process count in real-time...
- Ability to send HTTP request to the server with unique key, RAM, CPU and PROCESS.
- Ability to update client resource parameters to server via HTTP (REST) every 5 minutes.
- Ability to log resource parameter send and errors occurred to a local log file.

Server App:

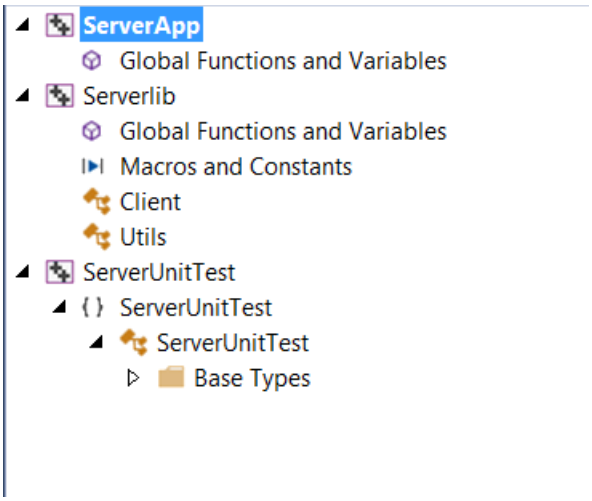
- Ability to load config xml with client alert generation parameters.
- Ability to update config.xml parameter to the system without restarting the system.
- Ability to use email template for alert mail.
- Ability to change email template without restarting the system.
- Ability to parse REST request data to client key, RAM, CPU and PROCESS parameters.
- Ability to log REST request parameters to MSSQL server.
- Ability to incorporate automated routing and notifications based on business rules specified in xml.
- Ability to generate alert for each client request.
- Ability to send separate alert mail for each violation to each client.
- Ability for each alert mail to understand what happened based on client key, email and resource parameters that are mentioned in the mail.

4. High Level Presentation of the Data Modal

- Client App
  - The ClientApp project consist of 3 projects
    - Client App : The entry point
    - Clientlib : Library having the Data model
    - ClientlibUnitTest: The Unit testing code for client lib

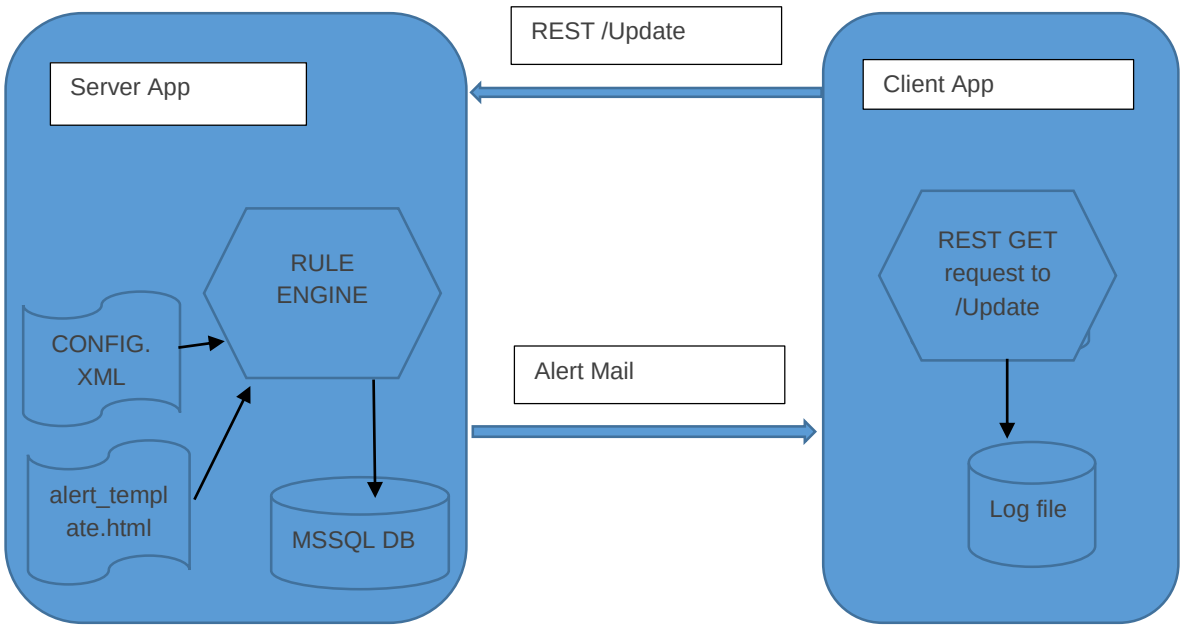


- Server App
  - The Server App consist of 3 projects
    - ServerApp : The entry point
    - Serverlib : Server data model and functions.
    - ServerUnitTest: Unit testing code for Serverlib



5. Architecture Diagram of the System

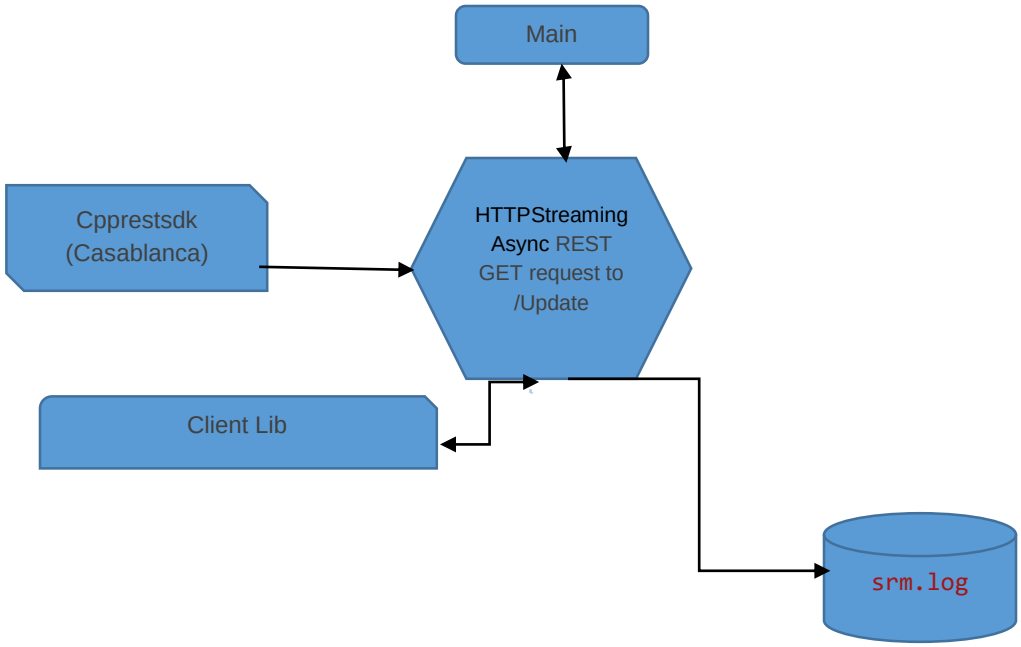
The following diagram Architecture Diagram of both client and server side together.



6. Architecture Breakdown of server and client

- Client App

Architecture Diagram of ClientApp



- Client App: does the HTTP/REST Update request to the server every 5 minutes. It calls clientlib .ClientUtils class to get the hostname, RAM, CPU, Process count and logs the data to a log file.

The client App main method gets the server IP and pass it to client lib .The client App calls the `HTTPStreamingAsync` function to send update request to http://serverip:8080/Update

The request gets the system resource details from Clientlib and formats the URL and does a REST GET request using cpprestsdk (Casablanca). After that client lib is used to log the data to a local file

- ClientLib:

  Client lib does all the logic part. The business logic resides here. Each function in the client lib and its explanation is mentioned below.

  Private Functions:

Function	Description
`float CalculateCPULoad(unsigned long idleTicks, unsigned long totalTicks)`	Calculate the CPU load.
`unsigned long FileTimeToInt64(const FILETIME & ft);`	Calculate the FileTimeToInt64.

  Public Function:

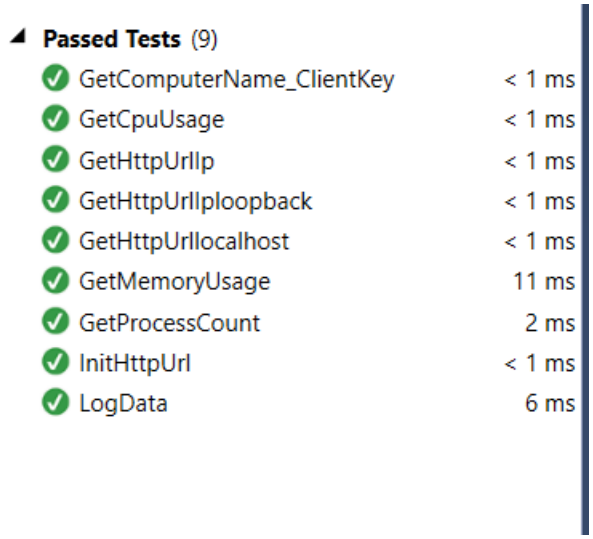
Function	Description
`ClientUtils()`	Constructor
`int InitHttpUrl(string serverIp);`	Converts server IP to http://serverIP:8080/Update and update the member variabel
`string getHttpUrl();`	Retrives the url for sending resource data. The url is processed in InitHttpUrl function
`string getHostName();`	This is the client key in the xml. This gives each pc in intranet a unique identification
`double GetMemoryUsage();`	The RAM usage is calculate using windows api and updated in member variable and returned
`double GetProcessCount();`	Get the number of process used, using windows api and updated in member variable and returned
`double GetCpuUsage();`	This gets the CPU usage by calling private function CalculateCPULoad and convert it to %

`int LogInfo();`	Logs the system resource parameters like hostname,RAM,CPU,Process usage to the log file
`int LogInfo(string logMessage);`	Logs the error message to the log file
`long getSleepInterval();`	Retrieve the 5 minute sleep interval calculated in milliseconds
`~ClientUtils();`	Destructor

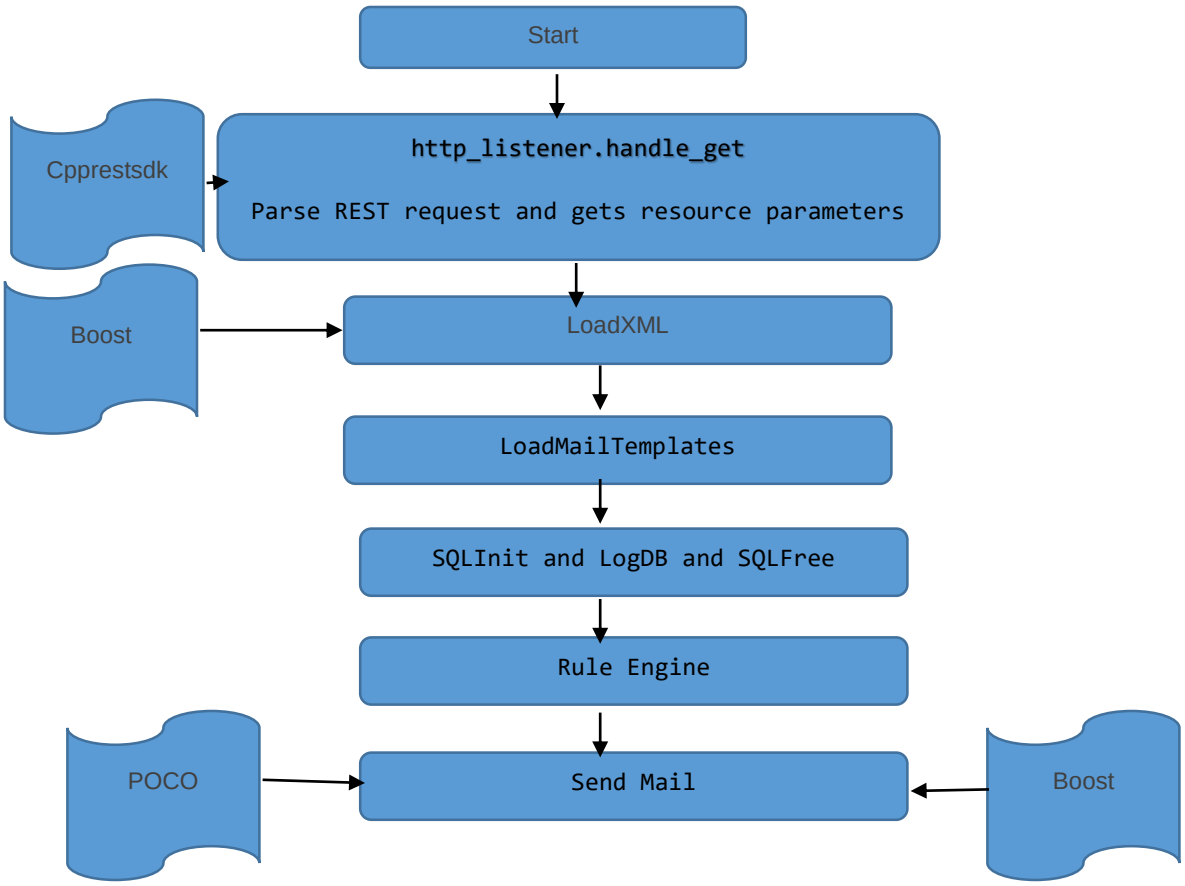
The class diagram of Client lib is mentioned below.

- ClientlibUnitTest:

  There are total of 9 test cases. This test all the class members of Clientlib. The class diagram, is displayed below for better understanding. The test case name is self-explanatory regarding the functions it is going to test.



- Server App
  - Architecture Diagram of ClientApp



- ServerApp

  The server app listens to HTTP request and runs a rule engine. HTTP/REST request are handled using cpprestsdk library. The server app listen to http://serverip:8080/Update

  Only REST listening is implemented in server app. The rest is implemented in server lib.

- Serverlib

  The serverlib does all business logic processing. These are functions is serverlib There is also a client object which is used to parse the xml. Server lib uses boost and POCO libraries.

  Boost is used for advanced string processing in formatting the email template and POCO is used to send mail. Boost is also used in xml parsing

The rule engine is in the client lib. Based on the rules defined in the xml the alerts are generated. When an alert is triggered a mail is send to the user. All parameters needed to trigger the alert are specified as a config.xml. Sample congig.xml is mentioned below.

<Server>

<client key="PC01" mail="test@gmail.com">

```
  <alert type="memory" limit="50%" />
  <alert type="cpu" limit="20%" />
  <alert type="processes" limit="50" />
</client>
<client key="PC2" mail=" test1@gmail.com">
  <alert type="memory" limit="50%" />
  <alert type="cpu" limit="20%" />
  <alert type="processes" limit="50" />
</client>
<client key="PC03" mail=" tes2t@gmail com">
  <alert type="memory" limit="50%" />
  <alert type="cpu" limit="20%" />
  <alert type="processes" limit="50" />
</client>
<client key="PC04" mail=" test1@gmail.com">
  <alert type="memory" limit="50%" />
  <alert type="cpu" limit="20%" />
  <alert type="processes" limit="50" />
</client>
</server>
```

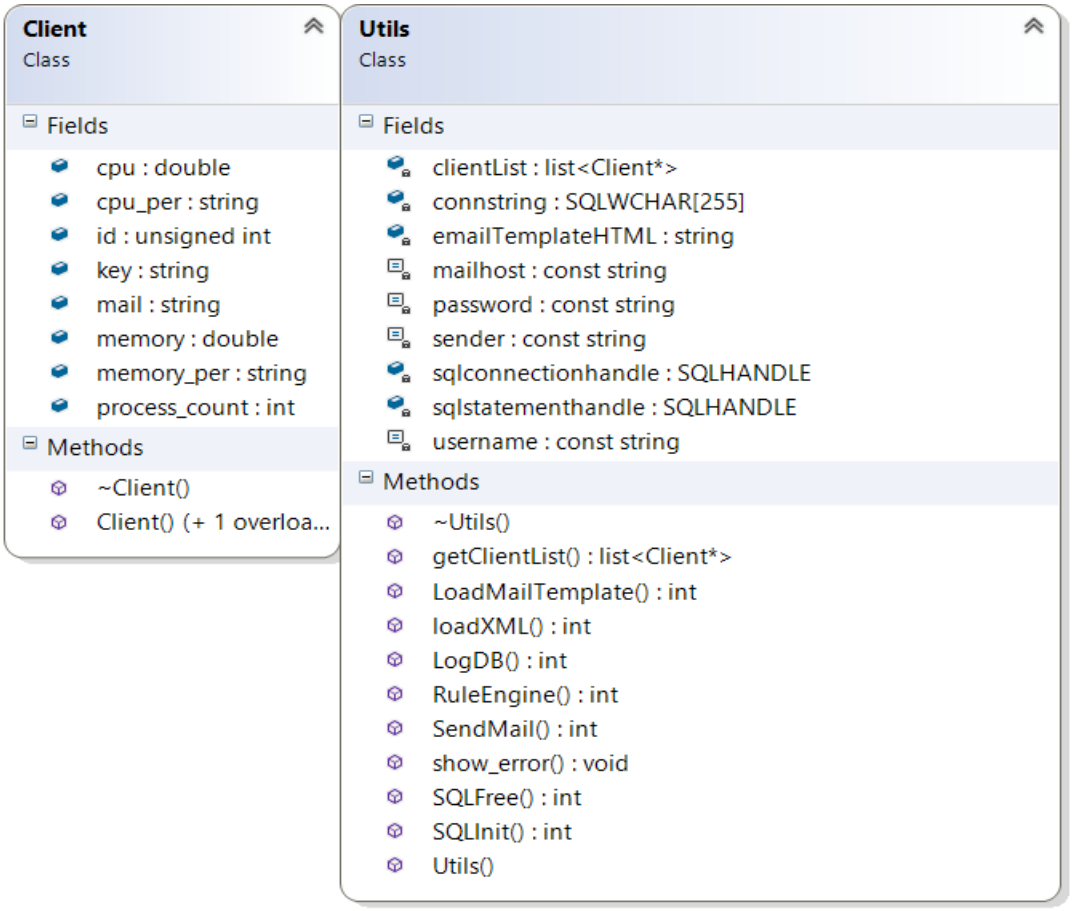
Each client app sends a REST request the data in the rest request are client key, memory usage in percentage, CPU usage in percentage and number of process running. The client key will be the PC name. Since the app is running in a LAN this helps to get a better identification of the pc when configuring the xml. Based on the client parameters alerts are checked. If any of the parameter condition is attained an alert is triggered. Based on the xml parameters and rest request parameters the alert html template is updated. This updated html is send as a mail to the pc user.

After that each REST request received is logged to the MSSQL DB. The REST interface is implemented using cpprestsdk. Boost library is used for Sending Mail. Boost does the porting easier. POCO does the SMTP part. The email template is formatted using boost library to update the system resource parameter.

Function	Description
`Utils();`	`Constructor`
`int loadXML(std::string filename);`	`Loads the xml and parse it and fill it to client list. This has`

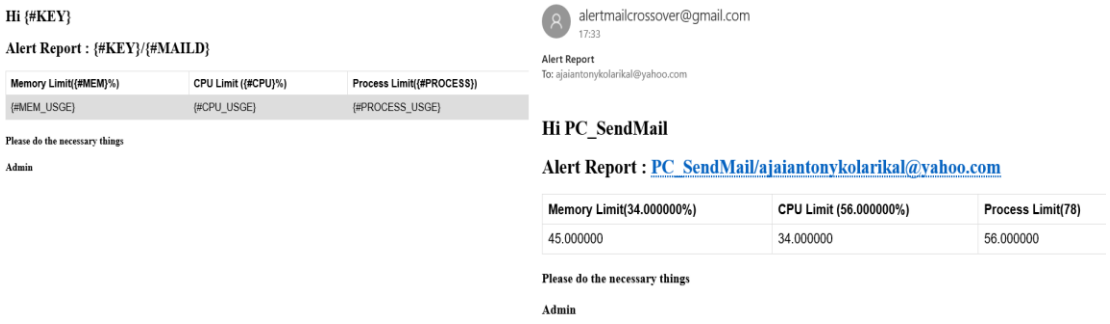
	the list of clients that are monitored. Their hostname , email to be used for sending the mail and the limit parameters
`std::list< Client*> getClientList();`	Get the list representation of the XML
`int SQLInit();`	Initialize the sql variable
`void show_error(unsigned int handletype, const SQLHANDLE& handle);`	Display the SQL handlers value
`int LogDB(string clientkey, double mem_usage, double cpu_usage, double processcount);`	Insert the client parameters to the DB
`int SQLFree();`	Free the DB handlers
`int RuleEngine(string strkey, double ram, double cpu, int process);`	This checks for violation or match of client value with xml values. If there is a condition triggered a mail is send And a console is logged.
`int LoadMailTemplate(string emailTemplate);`	Load the email template to the string
`int SendMail(string strkey, string recipientemail, double dramusage, double dcpuusage, double dprocessusage, double ramusagelmt, double cpuusagelmt, int processusagelmt);`	Send mail parses the email template string, fill it with data and sends it. The email provider data is stored in constants private

Class Diagram:



Email Template:

The email template is displayed below. Left hand the original template and right side the procesed Mail.



Hi {#KEY}

Alert Report : {#KEY}/{#MAILD}

Memory Limit({#MEM}%)	CPU Limit ({#CPU}%)	Process Limit({#PROCESS})
{#MEM_USGE}	{#CPU_USGE}	{#PROCESS_USGE}

Please do the necessary things

Admin

alertmailcrossover@gmail.com
17:33

Alert Report
To: ajaiantonykolarikal@yahoo.com

Hi PC_SendMail

Alert Report : PC_SendMail/ajaiantonykolarikal@yahoo.com

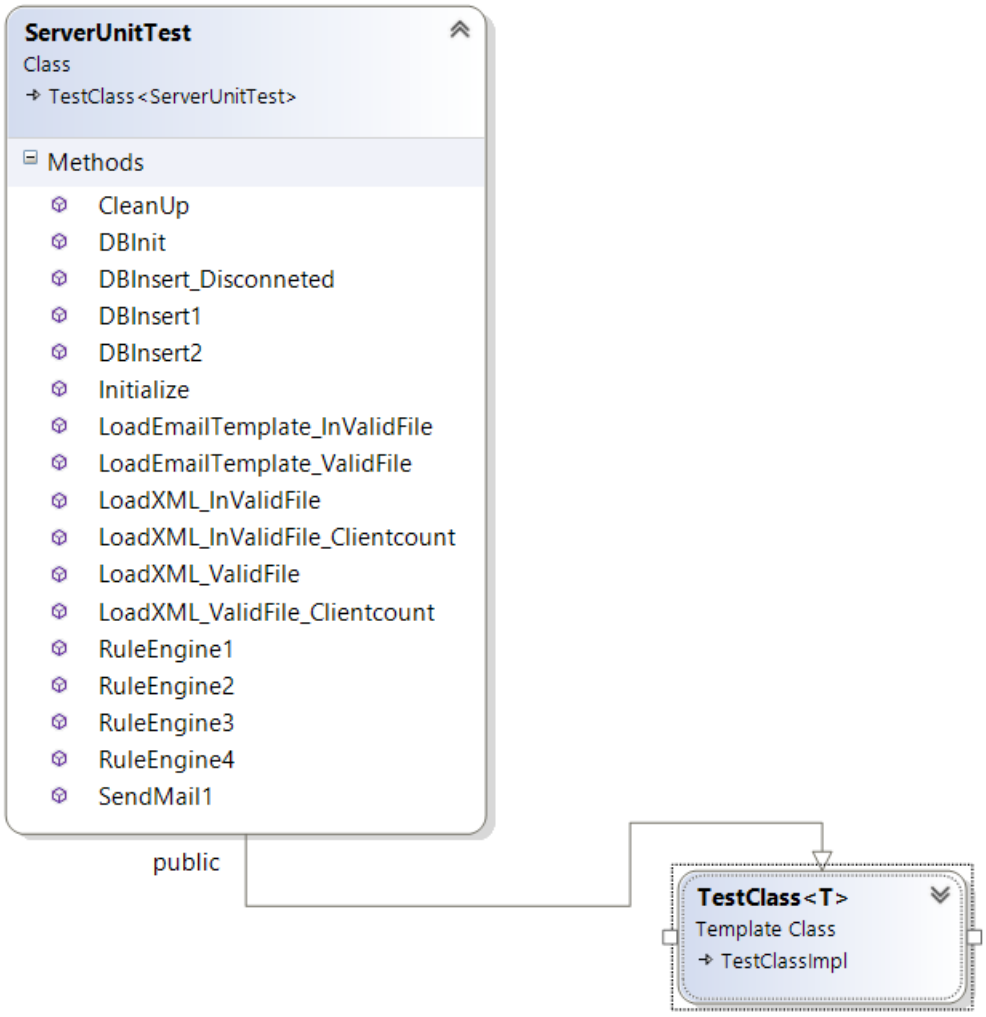
Memory Limit(34.000000%)	CPU Limit (56.000000%)	Process Limit(78)
45.000000	34.000000	56.000000

Please do the necessary things

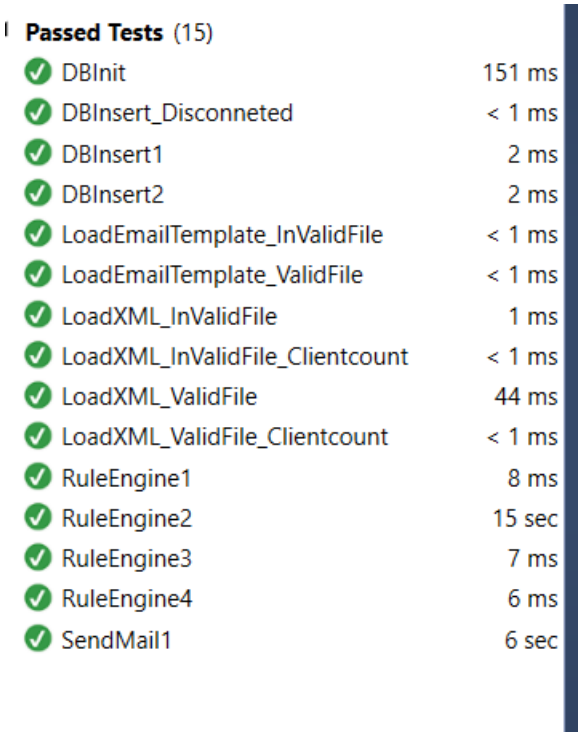
Admin

- ServerUnittest

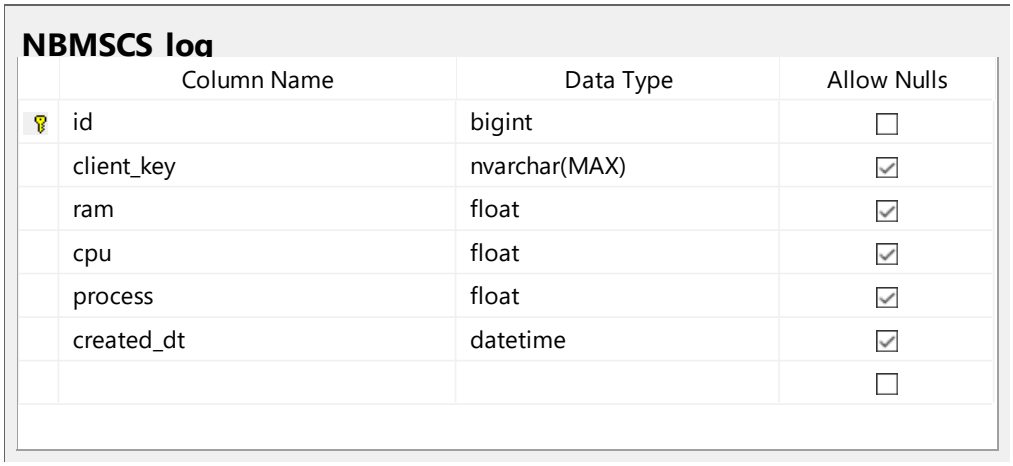
  The class diagram explains the list of unit test. The names are self-explanatory



There are a total of 15 test cases executed



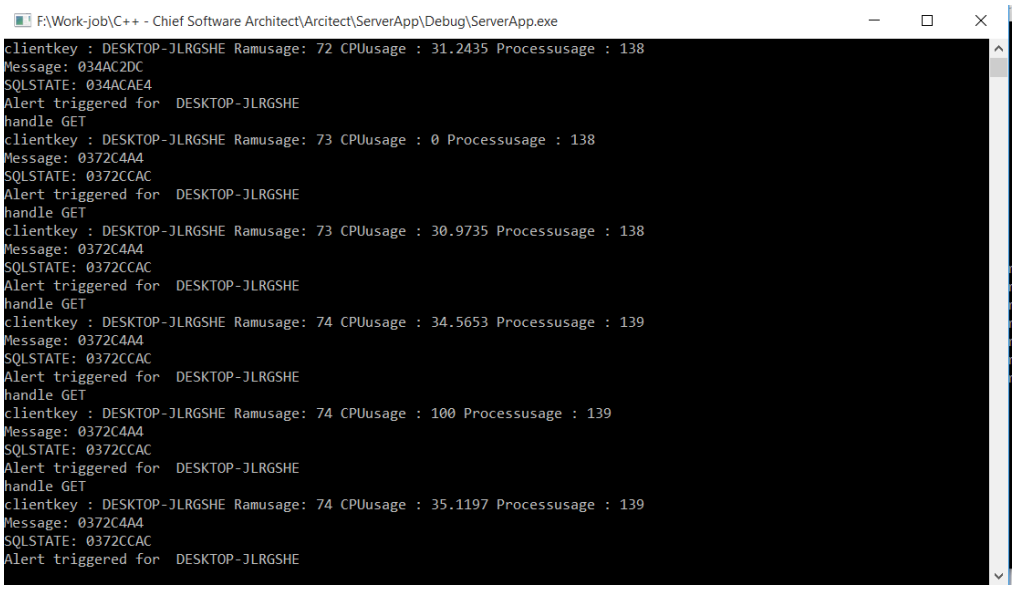
Ther Server DB table class diagram is shown below.



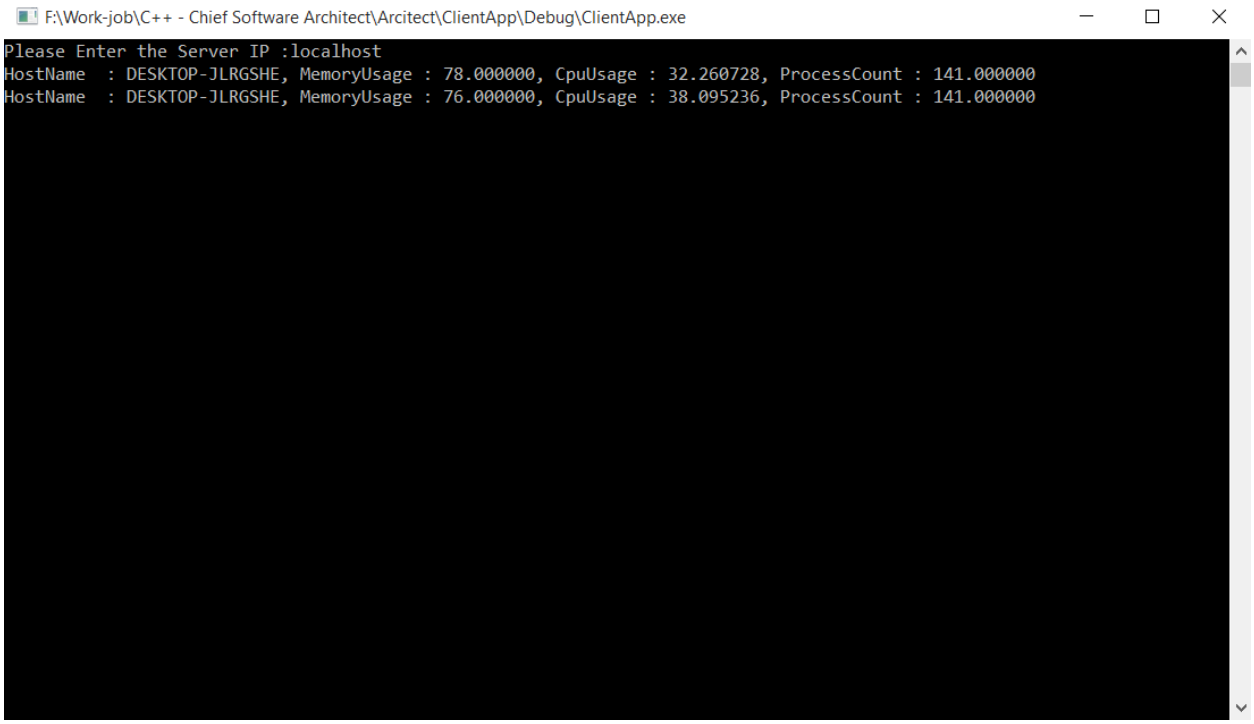
The received massages are logged in to table with timestamp. So it will easy for later analysis.

7. Sample Output

Server App:



Client App:



8. Specific Exclusions from Scope

NILL