# C:\Users\Reaper\Dropbox\westminster\YearTwo\Semester1\ecsc503.y-SoftwareDevelopmentGroupProject\uow-gp-1\Logo\GroupLogo.pngECSC503.Y

# Software Development Group Project

# Performance Monitor

# Group 2 Report

Contents

[ECSC503.Y – Software Development Group Project - Performance Monitor - Group 2 Report 1](#_Toc436310056)

[Problem scope 3](#_Toc436310057)

[Aims 3](#_Toc436310058)

[Assumptions of the Problem Domain and Problem Scope 3](#_Toc436310059)

[Context Diagram 4](#_Toc436310060)

[Use Case Diagram 5](#_Toc436310061)

[Use Case Scenarios 6](#_Toc436310062)

[Situation Flow 6](#_Toc436310063)

[Basic Flow 6](#_Toc436310064)

[Alternative Flow 6](#_Toc436310065)

[Exception 7](#_Toc436310066)

[Post Conditions 7](#_Toc436310067)

[Negative Situations 7](#_Toc436310068)

[Exceptions 7](#_Toc436310069)

[Intentional Threats 7](#_Toc436310070)

[Unwanted Scenarios 7](#_Toc436310071)

[Sequence Diagrams 8](#_Toc436310072)

[Standard User - Sequence Diagram 8](#_Toc436310073)

[Advanced User – Sequence Diagram 9](#_Toc436310074)

[Functional Requirements 10](#_Toc436310075)

[Standard 10](#_Toc436310076)

[Admin 10](#_Toc436310077)

[Class Diagram 11](#_Toc436310078)

[Relation of the Class diagram 12](#_Toc436310079)

[Inclusion of the Classes in the Design 12](#_Toc436310080)

# Problem scope

This report is to detail how our group came to include the aspects in our Network/ Performance Monitor.

## Aims

The aim of our project is to provide a stable and efficient interface for standard/administrative users to monitor their system performance. Within the report, we will highlight how the system will be developed and what we have taken into consideration. The diagrams we create will specify the interactions between multiple components of the system and the users.

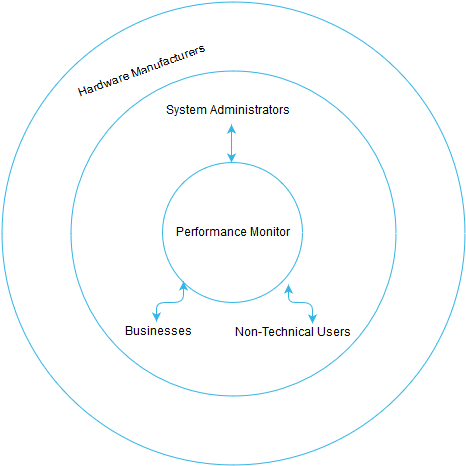
## Assumptions of the Problem Domain

The stakeholders currently use a set of tools to monitor how their system(s) are performing. Using multiple tools is taxing on the system resources, contributing to performance issues. We have come to the conclusion that this is not an ideal solution.

* Resource intensive
* Statistics overlapping – inaccurate/ outdated information
* Inefficient time management
* Poor team management
* Low security – anyone can access it

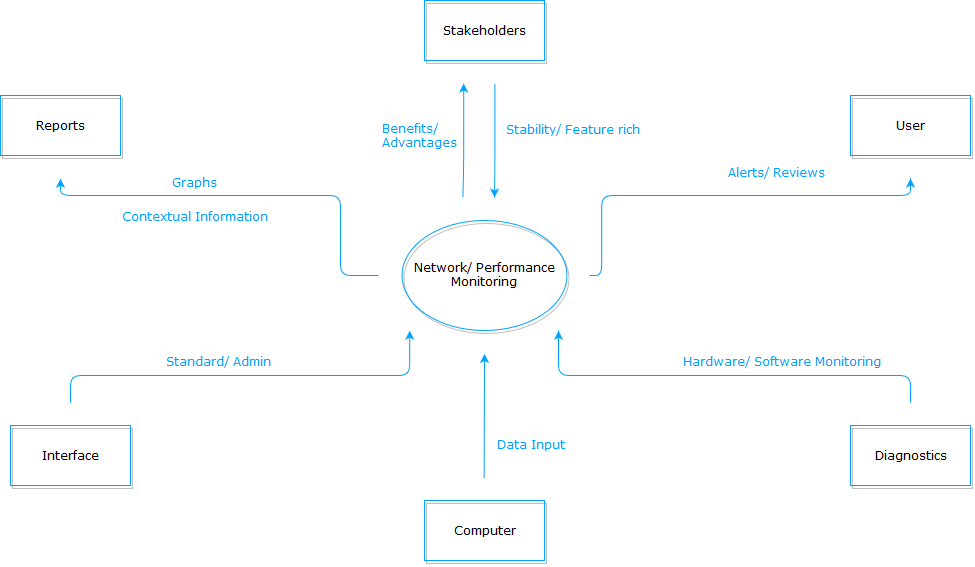
With these issues, it is clear that the stakeholders are looking for a solution that can be managed easily and is secure. The monetary costs will be low as they do not need to licence multiple tools to monitor their system(s).

In order to tackle the multiple problems that the current solution possess; the developers (we) have considered many improvements from our design of the software. The software can now be managed by one user, who either authenticates as a standard or administrative user. The entire interface has been designed to house all the information that a user may need, and also include a feedback system.



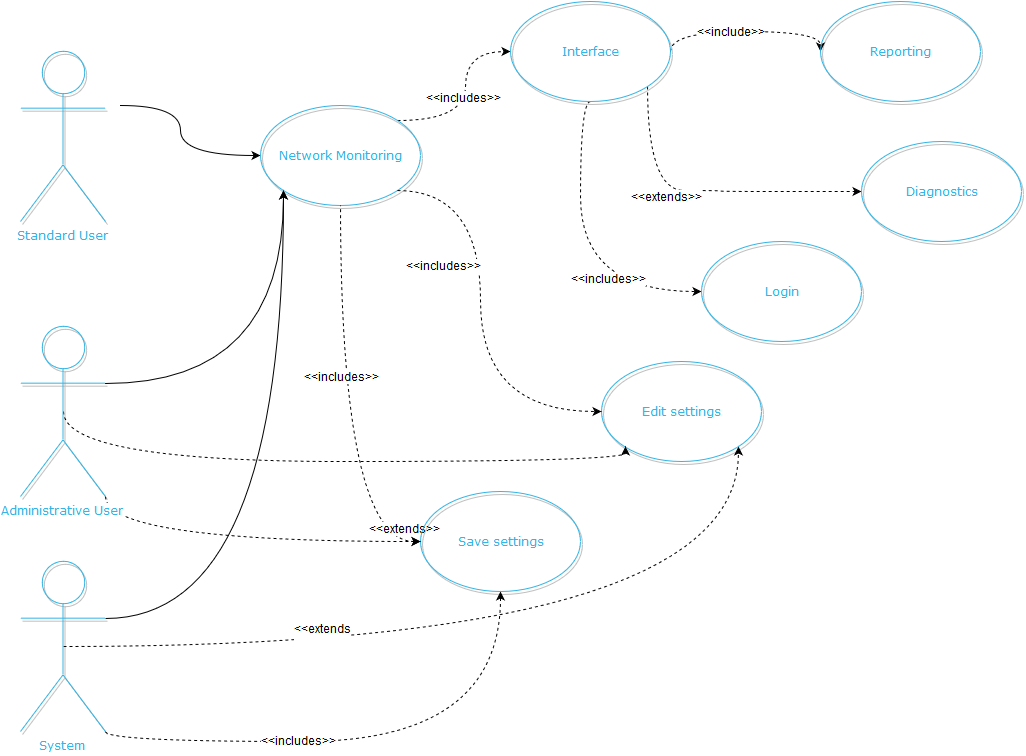
This diagram is a scope which represents the program. The inner circle represents direct interaction with the program, the outer circle represents indirect influence of the program.

# Context Diagram



The context diagram will allow for us to view our system of interest and its interactions with the environment. Arrows going towards the centre indicate that the environment is interacting with the system of interest; this could be attributes such as specific hardware monitoring, user input or the GUI displayed. Arrows pointing outwards from the centre indicate that the system of interest is outputting specific attributes. For example, in order to generate graphs, the program must out put the data to the corresponding terminator (Reports).

# Use Case Diagram



# Use Case Scenarios

Use case scenarios are beneficial for a team to evaluate what could potentially happen in a certain environment and what may happen when a problematic variable is introduced. The idea behind scenarios is to make the end product adaptable by using forethought in order to apply a solution for a situation that may occur.

Initially we considered who may want to use the program, this could be one of the following:

* Advanced technical user – System administrator
* Technically competent user – standard user
* Enterprises

We then questioned what the users may want to do with the program, as it is a performance monitor this can entail many different things that fall under the same scope. After some reflection, the group decided that these were the most important reasons as the why the program may be used and what it may be used for.

* Network Monitoring – monitor the throughput of the network and any potential traffic problems
* System Health – general performance overview, CPU usage, System uptime etc.
* Troubleshooting – check link states, possible system issues such as RAM usage being too high
* Reporting/ Reviews/ Alerts – User notifications (implemented to admin class users) utilised for troubleshooting
* Prevention of future problems – using performance monitor to precondition the system environment

## Situation Flow

Making a basic situation flow, we used the scenario that would entail a normal usage day where there would be no issues on either the programs side or the usage side.

### Basic Flow

**Use Case**: Performance Monitoring

**Description**: Advanced/Standard User wants to monitor the performance of the system

**Actors**: User, System

Preconditions:

1. Appropriate account is used; program has access to hardware monitoring sensors
2. User wants to see how system is performing.
3. System detects user input, authenticates user.
4. Depending on user type, appropriate interface is loaded for User (basic/advanced).
5. Standard user gets basic overview of system performance, Advanced user gets reporting, reviews and separate tab for network performance.
6. User logs off from program.

### Alternative Flow

In our alternative flow, we identified the issue of authenticating the user. If the user failed to authenticate themselves there needed to be a resolution and also a way to protect the system from misuse.

* Step 2 – Failure to authenticate correctly causes the program to terminate after three bad attempts.

## Exception

In an attempt for the user to regain access to the program, there were some solutions that we thought could be implemented.

User could restart the program after it terminates and use a ‘*reset password*’ button, the problem this poses is a reset is an easy system bypass. To remedy this the group proposed an idea:

If password is reset, user will enter answers to two/three security questions, and possibly a randomly generated alphanumeric string that must be entered.

## Post Conditions

The program should be able to identify critical changes from user input; if the user has changed their password but it was lost on the restart of the program, it is flawed. This issue reiterates when an advanced user changes some settings but they are not saved.

On user input, password is saved and then possibly backed up?

Admin user settings are saved whenever settings are modified.

## Negative Situations

Careful consideration of the use case, allows us to identify possible negative situations that could occur at certain crucial moments of the program’s development/ operation.

## Exceptions

Events occurring at normal operation of the program that would interrupt progress:

Application does not start or crashes during operation

Advanced/ Standard user views are not given to appropriate user i.e. Advanced user gets basic view

Program fails to retrieve system statistics

A password that is reset has not been saved, meaning that when the user enters the new password it is incorrect

Settings that are changed have not been saved

## Intentional Threats

Events that are caused by the action of hostile stakeholders or issues within the team:

Stakeholders reactively change desired features of the program

Changes are made to the program without members consulting each other

## Unwanted Scenarios

Events that are undesirable:

User correctly authenticates, but the program terminates on login or crashes

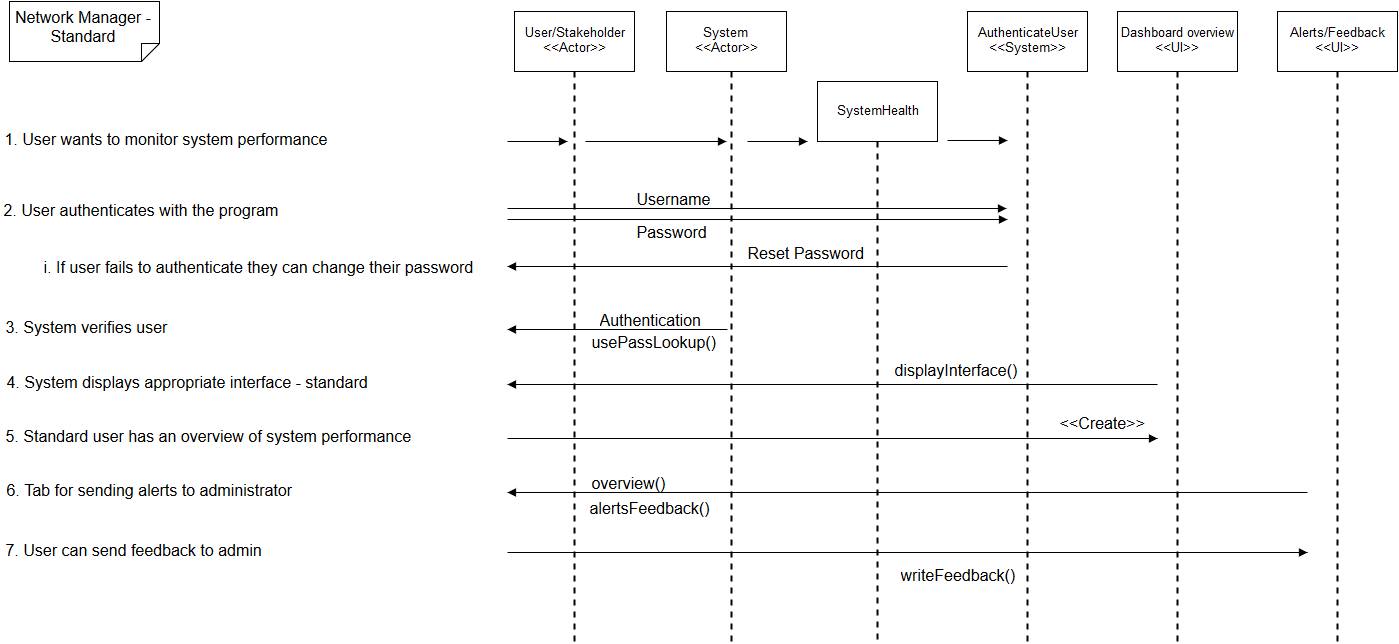
When the user fails to authenticate correctly, but the program does not terminate

The team does not deliver the program on time

The team fails to meet targets and/or delivers the program with missing essential features

# Sequence Diagrams

## Standard User - Sequence Diagram



This sequence diagram illustrates how the system and environment will operate with one another, for a standard user. The standard user will only need to see overall system performance and send feedback to the administrator.

## C:\Users\Reaper\AppData\Local\Microsoft\Windows\INetCache\Content.Word\SequenceDiagramAdministrator.pngAdvanced User – Sequence Diagram

Compared to the standard user sequence diagram; the Advanced User diagram has more operations for the different roles that an advanced user will assume.

# Functional Requirements

Functional requirements are ‘System shall do <task> here”. As opposed to non-functional, “where system will be <behaviour>” in a certain manner.

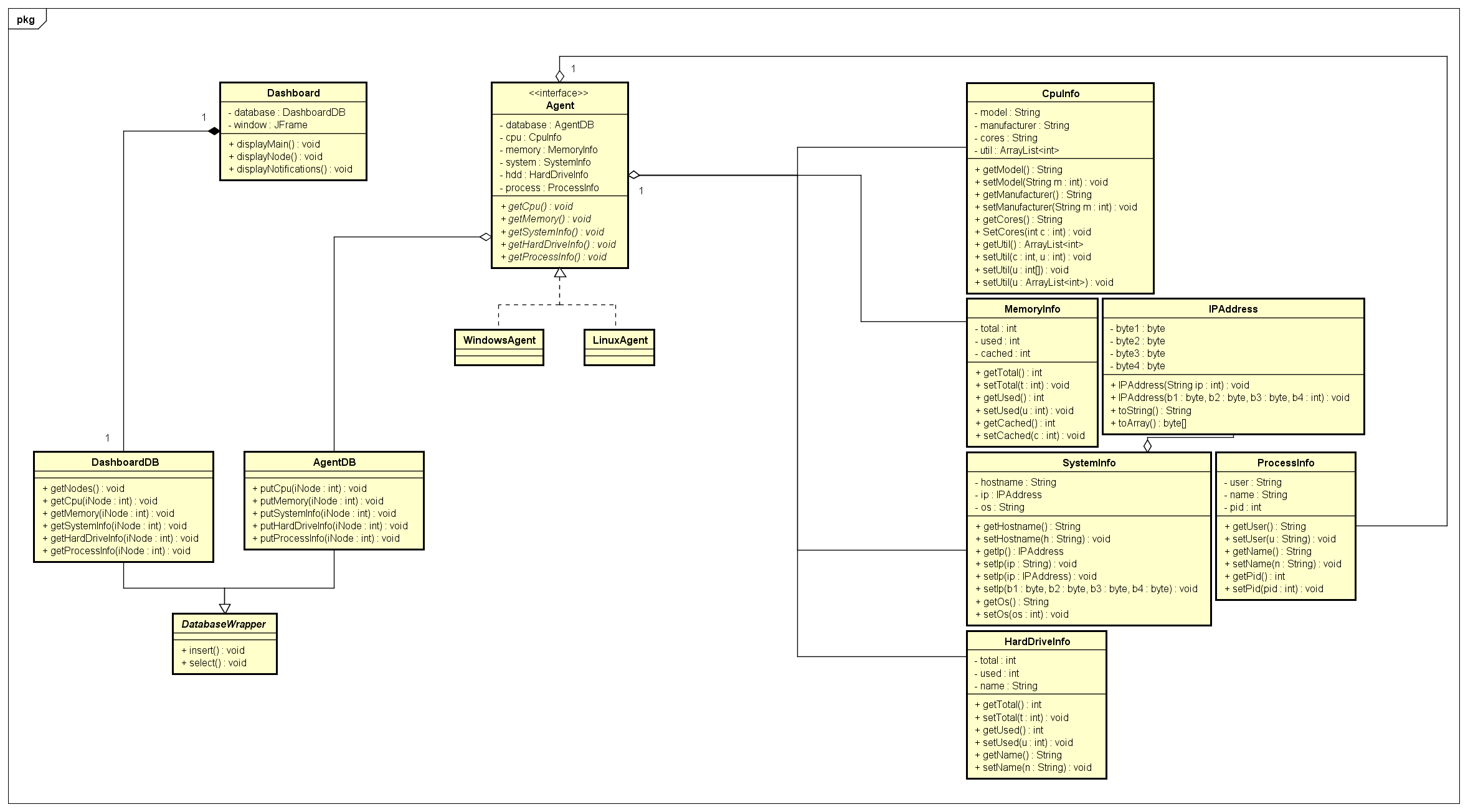
## Standard

* Users will have an overview of system performance.
* Users can send alerts to administrators.
* User is able to change forgotten password.

## Admin

* Administrative users will have views of individual performing components.
* Administrative users are able to make/ save changes.
* Administrative users can modify/ add/ remove, users respectively.
* ­­Administrative users have access to reports.
* Administrative users can see alerts from other administrators and standard users.

# Class Diagram



## Relation of the Class diagram

The class diagram, in relevancy to the sequence diagram, shows how the objects of the program will interact with one another. Some of the objects are grouped as they fall under a similar enough category.

## Inclusion of the Classes in the Design

Our class diagram represents the implementation baseline for classes used throughout our application. Considering we’re splitting our program into 2 separate executable parts, our class diagram contains 2 ‘top-level’ items: Agent and Dashboard.

The Agent interface is an abstraction of the actual agent application, and specifies which methods the OS-specific versions should contain (what data they should gather). Currently, we are planning on rolling out a Windows version, and if time permits a Linux one. We will be using a few helper classes to store the retrieved data for easier manipulation, this includes classes for IP addresses, system information, hard drive information, CPU usage and status, data about current memory state and process details.

The second top-level item is the Dashboard. This will be a separate program used to actually display the gathered data. Its structure isn’t absolutely clear at this point as we have multiple third-party libraries and various designs on the table.

Both the Agents and the Dashboard will use a database wrapper, each a customized one for its use case. This means that Dashboard’s database wrapper will contain methods mostly for retrieving data, while the Agent’s one will contain mainly data-inserting ones.