Contents

[A. Definition, Investigation and Analysis 3](#_Toc439921003)

[A. i) Definition 3](#_Toc439921004)

[The Problem 3](#_Toc439921005)

[Correspondence 1 Transcript 3](#_Toc439921006)

[Key Points 4](#_Toc439921007)

[End Users 4](#_Toc439921008)

[Data 4](#_Toc439921009)

[A. ii) Investigation and Analysis 7](#_Toc439921010)

[Methods of Data Gathering 7](#_Toc439921011)

[Integration with existing systems 12](#_Toc439921012)

[Similar Systems 13](#_Toc439921013)

[Possible Solutions 15](#_Toc439921014)

[Decided Solution 15](#_Toc439921015)

[Investigation into Hardware Requirements 16](#_Toc439921016)

[Requirements Specification 19](#_Toc439921017)

[Hardware & Software Requirements 21](#_Toc439921018)

[B. Design 22](#_Toc439921019)

[B. i) Nature of the solution 22](#_Toc439921020)

[Explanation of the Traditional SDLC Process 22](#_Toc439921021)

[Systems Diagram 25](#_Toc439921022)

[Directory Structure 31](#_Toc439921023)

[MySQL Database Structure 32](#_Toc439921024)

[Overarching Interface Designs 35](#_Toc439921025)

[User Interface Designs 38](#_Toc439921026)

[Admin Interface Designs 42](#_Toc439921027)

[User Sign off Design Proposal 52](#_Toc439921028)

[B. ii) Algorithms 53](#_Toc439921029)

[Database 53](#_Toc439921030)

[Leaderboard 55](#_Toc439921031)

[C. iii) Test Strategy 57](#_Toc439921032)

[Test Plan 57](#_Toc439921033)

[C. Software Development and Testing 58](#_Toc439921034)

[Initial class setup and database queries 58](#_Toc439921035)

[27th November 2015 58](#_Toc439921036)

[1st December 2015 67](#_Toc439921037)

[7th December 2015 69](#_Toc439921038)

[Website work 73](#_Toc439921039)

[10th – 15th December 2015 73](#_Toc439921040)

[D. Documentation 88](#_Toc439921041)

[E. Evaluation 89](#_Toc439921042)

[E. i) Discussion of the degree of success in meeting the original objectives 89](#_Toc439921043)

[E. ii) Evaluate the user’s response to the system 90](#_Toc439921044)

[E. iii) Desirable Extensions 91](#_Toc439921045)

[Project Appendixes 92](#_Toc439921046)

# Definition, Investigation and Analysis

## A. i) Definition

### The Problem

With each coming generation it is becoming harder to get students in a school setting to be interested in working to the best of their ability. With computer games now offering tangible rewards for putting in effort, the feeling of success without physical or even virtual achievements, titles or items is becoming lower. The solution? Integrate solutions found in many modern computer games into a real world student learning environment, where students can work to earn achievements, and reap the rewards for doing so.

My client, Mr. Craig Sargent, head of ICT & Computing at Archway School would like such a system turned into a fully automated, programmed application, which allows the distribution of achievements after proof of work, without much effort from either him, or the students.

### Correspondence 1 Transcript

I asked some questions to my client over e-mail. This is the transcript of that conversation.

**Q1: What is the general purpose of the system?**

To promote competition and fun, to increase the chances that students will engage in the learning processes.

**Q2: What input devices will the system be limited to? (Including in administration mode)**

Not sure what you mean?  If you literally mean traditional input devices, I intend this to be filled out online via a standard desktop computer, so keyboard and mouse.

**Q3: What kind of user interface should it use?**

Ideally web based, hosted either internally or externally so that students can access the system as well as staff from any location easily with internet access.

**Q4: How important are flashy graphics/visuals in the application?**

Not as important as getting the system working!  However the achievement system does have a very specific look I am going for in terms of colours, I can provide these to you.

**Q5: Since there will be multiple users, in what way will the users be able to interact with one-another?**

Probably very little at first.  Although it would be nice for users to be able to “compare achievements” with each other, or at least be able to view a leader board which dynamically updates.

**Q6: How would you want the teacher/admin mode to operate?**

The ability to create users, delete users, change user’s details etc.  Also be able to add/remove/alter all aspects of achievements.  Alter the number of levels in the systems, the ranks, the icons associated with those levels.  In other words to be able to full customise all aspects of the achievement system from one admin interface.

**Q7: How will the system encourage students to improve their work ethic?**

Hopefully by installing a sense of fun, excitement and competition like most achievement systems in most games.  The levels and ranks are going to have actual rewards associated with them

### Key Points

After correspondence with my client, the following points were raised:

* The application must be intuitive to use, with a simple, consistent and familiar interface.
* The achievement data will be stored on a networked database, so it can be accessed by multiple clients inside and out of the school site.
* Each achievement will be assigned a number of points, name, icon, and these values will be able to be modified to as seen fit as well as achievements being available to be added and removed.
* There will be an administrator version of the program which allows distribution of achievements, and management of users, user data and achievements.
* The application will automatically obtain the users ID when logging on a school system, otherwise, credentials will be requested.
* A leaderboard and other inter-user features will be available to install competition amongst the students.

### End Users

The end users of the application will be students taking the new A-level computing course. Aged 16-18, and the client, Craig Sargent, who will be using the administrator mode to distribute the achievements after verifying the requirements.

### Data

The database that will be storing the data for the application will do so through several tables. The contents of a single record on each table is displayed below.

#### Achievement table

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Data Type** | **Source** | **Use in Application** |
| ID | Integer | Auto-generated, unique identifier, primary key | To cross-reference achievements in other tables and application |
| Name | String | Entered in via the administration mode | The name of the achievement that will be displayed to the user. |
| Description | String | Entered in via the administration mode | The description of the achievement and how to achieve it that will be displayed to the user. |
| Image | String | Entered in via the administration mode | The HTML link of the image being used. This is to reduce the overhead of the application, make the images dynamic, and to allow the image to be accessed from on and off the school site. |
| Value | Integer | Entered in via the administration mode | The score that the achievement is worth. |
| Date Added | DateTime | Automatically entered by the program when the achievement is created. | Information available to the administrator that may be of use. |
| Hidden | Boolean | Entered in via the administration mode | Used by the application to determine whether or not the achievement description will be displayed before the achievement is obtained. |

#### User table

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Data Type** | **Source** | **Use in Application** |
| ID | Integer | Auto-generated, unique identifier, primary key | To cross reference the user in other tables and the application. |
| Username | String | Created by the application after recognising the network as Archway, and the user selecting create new user. | To give a name to each user that is the same as the school’s system usernames. Allows the application to recognise the user if they are logged into the school system. |
| Real Name | String | Entered by either the application by pulling it from the school system, or, if this is not possible, by the administrator. | To allow the administrator to easily identify which user belongs to who. |
| Nickname | String | Entered by the user when they first create their user and can be edited at a later date. | To allow each user to set their own personal username that will be used in inter-user leaderboards and suchlike. |
| Total Score | Integer | Automatically created by the application upon creating a user, and updated upon achievements being obtained. | The overall score from the sum of all of the achievements that the user has obtained. |
| Rank | Integer | Automatically calculated based on the score. | Points to the ID of the rank that the user currently has. |
| Nick Locked | Boolean | Default false when the user is created, and can be changed via the administrator mode. | Allows modification of the nickname when false. Otherwise prevents the user from modifying their nickname in the case of abusive nicknames. |

#### Ranks Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Data Type** | **Source** | **Use in Application** |
| ID | String | Auto-generated, unique identifier, primary key | To cross reference the rank in other tables and the application. |
| Name | String | Entered in via the administration mode | The name of the rank to display to the user. |
| Required Score | Integer | Entered in via the administration mode | The minimum score required to achieve the rank. Used to calculate which rank the user has obtained. |
| Icon | String | Entered in via the administration mode | The HTML link of the image being used. This is to reduce the overhead of the application, make the images dynamic, and to allow the image to be accessed from on and off the school site. |

#### User Achievement table

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Data Type** | **Source** | **Use in Application** |
| ID | Integer | Auto-generated, unique identifier, primary key | To give each of the user-achievement link a unique identifier. |
| User | Integer | Created by the application upon distribution of an achievement to a user. | Links to the user that the achievement has been distributed to. The user’s ID. |
| Achievement | Integer | Created by the application upon distribution of an achievement to a user. | Links to the achievement that has been distributed. The achievement’s ID. |

## A. ii) Investigation and Analysis

### Methods of Data Gathering

#### Observation

By observing the current system in action, I should be able to pick up some of the key points that the system needs to accomplish. If the computerised system does not perform at least the functionality that was previously available, than the project is pointless. The observation will include me watching the current system in action by both the client, and his pupils that the system has been created for. This allows me to see the most frequently required functions, and get a feel for what the user interface should look like and be capable of.

#### Interviews

Interviews are an incredibly effective way of gaining information, as it allows responses based on answers to previous questions. This allows a greater range of data to be gathered from the client, especially when the data does not necessarily need to be gathered from a wide range of people. As interviewing is a fairly slow process, as the number of clients using the final system increases, its effectivity decreases.

Further interviews should give me access to critical information about the current system and the requirements of the new system. By interviewing the client directly, I can gather specific requirements that the system must be able to perform, that the current system does not offer. While the interviews will likely span a relatively short space of time, as only one client will be interviewed, time is not an issue. In this specific project, interviews will be my main source of data.

#### Questionnaires

Questionnaires are a fast effective way of gathering data from a target demographic that is a group. They are a cheap, effective alternative to interviews when needing to gather data from a range of people, but will not necessarily produce the same results, and cannot perform the same dynamic in-depth questioning that interviews can.

Due to the fact that the majority of the data that I shall be gathering will be from a single client and not a group, I am unlikely to use questionnaires. The time taken to produce them and distribute them could be more wisely used for further interviews and/or observation, which will gather more specific, in-depth requirements.

#### The current system

It’s important to review the current system that is being used, as the new system that is being produced needs to replicate all of the functionality of the previous system, and either make it more efficient, or add additional functionality.

The current system comprises of simply an Excel Spreadsheet to show which student has obtained an achievement, and a webpage with all of the relevant information about the achievement system on it. While this system works, it can be a laborious task when needing to update a large quantity of pupils, and is not very engaging for the students to see where they have achieved. A computerised system will be able to help with both of these, by making it easy to distribute the achievements to multiple students at once and making a Graphical User Interface that the student can really get engaged in due to fancy graphics and possibly slick animation, as well as a leaderboard. I can however, pull graphical assets from these systems to use in the final system, since all of the data that is required for the system will be pre-loaded, however it will be possible to change and remove this data.

#### Correspondence 2 Plan

I plan to ask the following questions to my client, and any follow-up questions that result.

What functions should the typical user have access to?

You mentioned a colour scheme previously. What is it?

What screens would you imagine that the student users should have?

How will the achievements be displayed?

How will the user be notified of new achievements?

What information will be available on the leaderboard?

What information should always be present to a user and available at a glance?

#### Correspondence 2 Transcript

**Q1: What functions should the typical user have access to?**

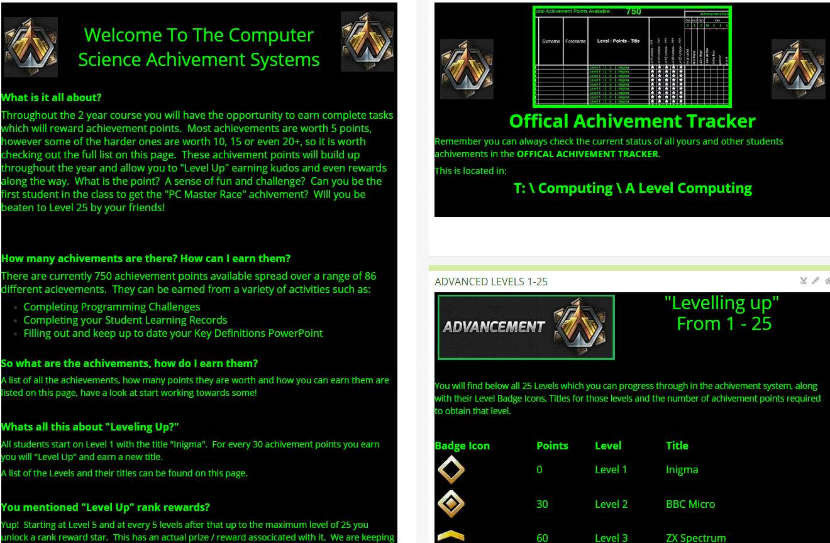
At this stage I am not sure, they need at very simplest level to be able to view their achievements and those of others

It would be nice to think they can “log-in” to the system, this implies they have the ability to change or request a password reset to their email address, so they should be able to “register” to the system as a “student user”.

If we get as far as them logging in it would be nice if they could “Request” an achievement which the teacher will then validate.  So they could log in and tick say 5 different achievements for the teacher to check, until the teacher has done so they see something like “You have the following achievements waiting to be validate by your teacher…”

**Q2: You mentioned a colour scheme previously. What is it?**

I have added you to my AS Computing course on the VLE <https://archway.itslearning.com> , please go there and check the achievements system so you can see the colour scheme.  I have also attached my achievements tracker which I am currently doing in Excel, the colours need to be like this:

****

**Q3: What screens would you imagine that the student users should have?**

You tell me  Give me some design.  They want to be able to see all the information that’s currently on the VLE page I gave you, plus they need to be able to see their achievements / compare to others / also maybe to be able to submit new achivements for validation and account admin (reset password)

**Q4: How will the achievements be displayed?**

Don’t really understand the question.  That is up to you really. Something like this?  Though I am willing to accept other solutions.  I def want achievements to be groupable by category (Programming, Student learning records, Key terminology, Meta achivements) – those are the 4 catagories I want from the start, I need to ability to add and remove achievement catagories



**Q5: How will the user be notified of new achievements?**

Again that’s up to you.  Simplest way is they just log on and see them.  Better solution is once the teacher has validated them they get an email “Congratulation your new achievements have been validated, you now have 40 achievement points and are level 4, log in to the system at www…. To see full details”

**Q6: What information will be available on the leaderboard?**

Name, Level, Rank, Title, Total achievement points (See achievement tracker – attached)

**Q7: What information should always be present to a user and available at a glance?**

Name, Level, Rank, Title, Total achievement points (See achievement tracker – attached)

#### Correspondence 1 Analysis

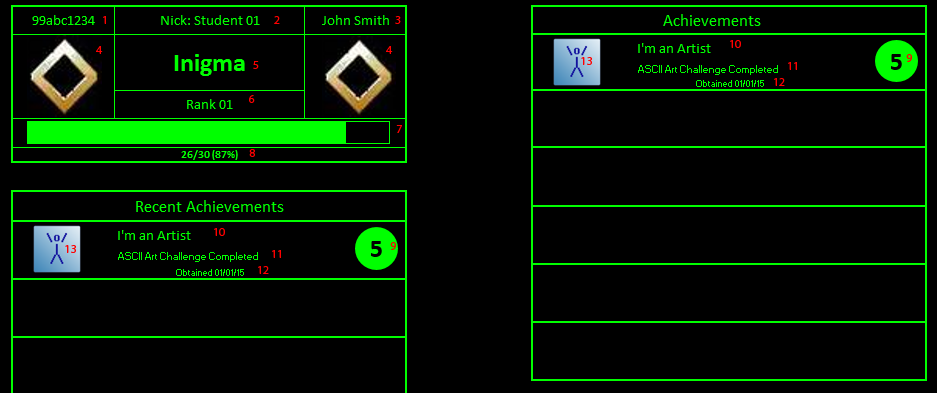
After looking through my first proper correspondence with the end user, the following points have been raised:

* Users can login to the system and access the following information and functions
  + View current achievements
  + View current Rank & score
  + Request a password reset
  + Change email preferences
    - Allow emails for new achievements being given
  + Request achievements that the user thinks they have achieved to be reviewed
  + Access the leaderboard
* The system will need to be able to achieve the following things:
  + Calculate when the user has enough points to go over a rank boundary.
  + Offer an interface to the user that allows rudimentary access to achievement data and user preferences
  + Offer a comprehensive interface to the admin that allows modification of the users, achievements, achievement categories and ranks
  + Send out e-mails to the user for password resets and to tell them when an achievement has been validated.
* Relatively simple student interface and comprehensive admin interface
  + Green and black colour scheme
  + Preferably web-based as this allows any user to login easily from anywhere with any device that has an internet connection.

#### Interview 1 Plan

For my first interview, I plan to get information on functionality as well as UI design for the user. Since all the end user will see is the UI, it is imperative that it functions exactly as desired. The backend solutions holding the program together only have to be optimised enough to give the UI a seamless experience to the user. I have created some simple diagrams to show what ideas I already have for the user interface, so during this discussion, the design does not have to start from scratch.

#### Interview 1 Analysis



After the first interview, this mock-up interface for the student users was produced. The left section is the user overview page which confronts the user after they log in. The top section has information while the bottom part lists their latest three achievements. The right section is another page which shows off all of the users achievements, and will ideally be able to be filtered based on different aspects of the achievement as well as the part of the course the achievement is related to. The features have been numbered and are as follows:

1. The network/school username
2. The nickname of the user, set by the user
3. The real name of the user, either pulled from the school system or set by the administrator
4. The image of the rank that the user has achieved.
5. The name of the rank that the user has achieved.
6. The number rank of the rank that the user has achieved.
7. A progress bar showing how close the user is to the next rank only and updates upon reaching a new rank with a new maximum.
8. The information that the progress bar shows in a fraction and percentile format.
9. The number of points the achievement is worth.
10. The name of the achievement.
11. The description of the achievement.
12. The date the achievement was obtained.
13. The image associated with the achievement.

Other points that were raised in this interview were:

* The entirety of the student interface must be web-based, so the users can connect to it in and out of school and can be run on any machine with an internet connection and a modern browser (Not IE).
* The admin interface should be web-based, but an application solution could also work.
* There should be a leaderboard which displays the user’s nickname, rank and overall score. The leaderboard should have multiple filtering options including one to filter highest scores in each category of achievements.
* Each user should be able to change their nickname, registered e-mail address, e-mail preferences and password. If the nicknames become abusive, the administrator should have the right to set and lock nicknames. Password resets should also be available to the administrator.
* Users should be able to reset their password via email.
* The entirety of the data being used by the system including ranks, achievement categories, achievements, users, colour schemes and images should be modifiable by the admin. Object-orientation instead of direct references is a must.
* A rewards system should be available to the user. This allows the admin to set items that get presented to the user upon reaching a certain rank.
* The system will be pre-loaded before handing it to the user with his current system in its entirety, including achievements, ranks and rewards.

### Integration with existing systems

After a discussion with the IT manager, James Moth at Archway School, the location in which the system will be deployed, several key points were raised that will affect how the system is designed:

* The school has an internal webserver using Microsoft IIS (Internet Information Services) Express with its own domain. This will allow me to create and host web content which is likely to happen.
* The mail server that the school uses is Microsoft Exchange 2013 and this is hosted on-site. This is good as it means SMTP does not have to be used in order to send internal mail, but it will be required for external mail.
* The school has a MySQL database server on site that can be utilised, and this will be my chosen storage solution as it allows for easy access from a webserver and local applications using a library.

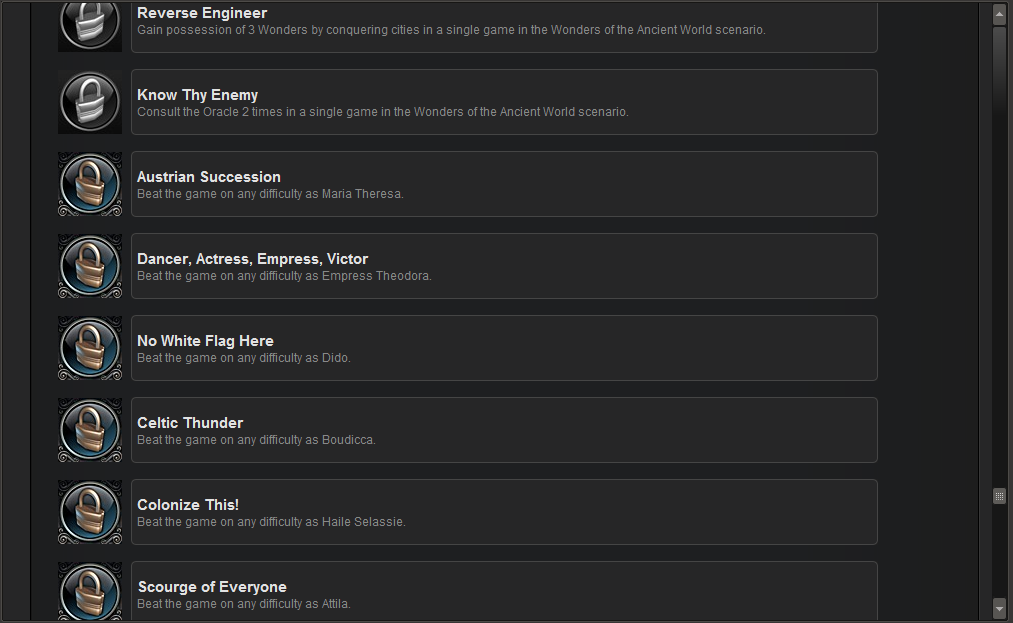
The IT manager is willing to give me access to my own database on the MySQL server to use for the project as well as a directory on the IIS webserver. Mailserver credentials can be sorted for external mail access at a later date.

### Similar Systems

Since the achievement system is being partially based on Games platforms such as Steam, Battle.Net and Xbox live, it would do justice to review these systems to see how they perform the tasks necessary.

#### Steam

Steam is an online games social platform and store for PC. It handles achievements for thousands of games in one single framework.



Above is an image taken from one specific game’s achievement screen. One thing that instantly strikes me is how simple the interface is, including the colour scheme. Because the colour scheme is consistent, this screen looks very clean, and makes it incredibly obvious where one element stops and another begins, on top of this, each element spans from one side of the screen to the other, with a single image representing the achievement; this adds to the clean look tremendously. Additionally, Steam doesn’t make comparing achievements with other users easy, making competition between users for achievements rare. It seems that Steam achievements are primarily just to give the user additional end goals, with no ranking or points system based off of it.

#### World of Warcraft

World of Warcraft is a Massively Multiplayer Online Role playing game (MMORPG) and has a comprehensive in-game achievement system that is solely for that title. 

The thing that instantly strikes me about this interface is its very well defined style. Every part of the UI in World of Warcraft is consistent to the house style with similar colours and themes. On top of that, this particular part of the UI has filters for various categories, something that has been requested of my system, and is using progress bars to show how many achievements have been completed out of the total number available. The entirety of the achievement points gained by the user is visible in every screen on this interface.

In terms of the achievement elements themselves, each one is similar to the style of steam, though there are slight differences. I will be basing my achievement elements off of these as it would do well for the style to feel consistent and familiar to user that have used these systems. The consistencies between my system and this system is the number of points the achievement is worth to the right side of the element, the name of the achievement, description and date achieved being visible, and an image associated with the achievement off to the right side.

### Possible Solutions

#### Web-based using MySQL, PHP

One possible solution in order to create the system would be to create all needed interface in PHP, and have a MySQL database backend. The PHP pages would query the database for all needed data. This would mean that the server would need to run a MySQL database, a webserver and a mailserver, and is the most lightweight option. The bad side of this solution is it would require that I learn PHP to a greater extent, making it a little less viable as it probably wouldn’t be as efficient.

#### Web-based using ASP.NET, MySQL

Another solution would be to use a web-based interface, but this time programmed in ASP.NET. This would mean that the elements would be easier to program for me personally as my native programming language is C#. It would also mean the same server setup as the web-based solution using PHP.

#### Application-based using .NET, C# & MySQL

This solution is the least optimal solution, as it would mean that all interfaces were application based, meaning that the applications would have to be deployed to all machines on the network, and downloaded if the user wished to login at home. This results in great inefficiency as well as hassle in deploying, as the firewall would also need to be configured. While this would be the easiest option for me to personally program, I cannot recommend it.

#### Hybrid using .NET, C#, MySQL & PHP

This solution is a little more complicated, instead of all interfaces being accessible in the same way, it would mean that the administrator interface would be a standalone application executable, and the interface for the users be web-based. What this would mean would be much greater access to functions for the administrator interface, as it would be able to pull things easily from the local network using .NET, and still allow the users to access the interface from anywhere. Unfortunately, the client would be happier if this wasn’t the solution, as web-based interfaces are preferred, so this solution is a no-go.

### Decided Solution

The final decided solution, due to the pros far outweighing the cons, is a web-based interface using ASP.NET, with a MySQL database backend, an IIS Express webserver and a Microsoft Exchange mailserver. The function that each of these will carry out is listed below:

* Web interface using ASP.NET
  + Provides the interface to the user and administrator
* MySQL Database
  + Provides dedicated, long-term storage that can be accessed by the web interface for all of the student records, settings and achievements that the administrator has created.
* IIS Express Webserver
  + Provides the server through which the web content will be displayed. Can accept ASP.NET classes.
* Microsoft Exchange mailserver
  + Provides a mailsever through which all e-mails that the system will send can be sent from.

### Investigation into Hardware Requirements

Since the application is not just simply an executable with some jpg’s thrown in, a complex analysis of what the requirements the hardware must adhere to is required.

#### Server – Database

A typical MySQL database with required libraries takes up about 210MB of space, with each table and records in the table taking up additional space. Each table will require about 16kB of space, the record size is dependent on the types that the record is storing.

**Record sizes**

Below are tables which show the likely contents of each table and therefore give an estimate for the record size. Using these should give me a pretty good estimate of the total size of the database.

|  |  |  |
| --- | --- | --- |
| **Achievements** | | |
| **Record Name** | **Type** | **Space (Bytes)** |
| ID | Integer | 8 |
| Name | String | 20 |
| Description | String | 200 |
| Image | String | 100 |
| Value | Integer | 8 |
| DateAdded | DateTime | 8 |
| Hidden | Bool | 1 |

Size of each record: 345 Bytes.

|  |  |  |
| --- | --- | --- |
| **Users** | | |
| **Record Name** | **Type** | **Space (Bytes)** |
| ID | Integer | 8 |
| Username | String | 20 |
| Password | String | 50 |
| RealName | String | 20 |
| NickName | String | 20 |
| Score | Integer | 8 |
| RankID | Integer | 8 |
| NickLocked | Bool | 1 |
| Disabled | Bool | 1 |

Size of each record: 136 Bytes.

|  |  |  |
| --- | --- | --- |
| **AchievementData** | | |
| **Record Name** | **Type** | **Space (Bytes)** |
| ID | Integer | 8 |
| UserID | Integer | 8 |
| AchievementID | Integer | 8 |
| Awarded | Boolean | 1 |

Size of each record: 25 Bytes.

|  |  |  |
| --- | --- | --- |
| **Ranks** | | |
| **Record Name** | **Type** | **Space (Bytes)** |
| ID | Integer | 8 |
| Name | String | 20 |
| MinScore | Integer | 8 |
| Image | String | 100 |

Size of each record: 136 Bytes.

|  |  |  |
| --- | --- | --- |
| **Rewards** | | |
| **Record Name** | **Type** | **Space (Bytes)** |
| ID | Integer | 8 |
| Name | String | 20 |
| Description | String | 200 |

Size of each record: 228 Bytes

So, with these estimates calculated, we can safely assume none of these tables will go past 1000 records. Some of them may become several hundred records in size, so this is a safe estimate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Bytes per record** | **X1000 (Bytes)** | **Size (MB)** |
| Users | 136 | 136000 | 0.13 |
| Achievements | 345 | 345000 | 0.33 |
| AchievementData | 25 | 25000 | 0.02 |
| Ranks | 136 | 136000 | 0.13 |
| Rewards | 228 | 228000 | 0.22 |

Total size of database with table data: 0.9MB + 10% overhead for OS = 0.99MB = 1MB.

Total amount of hard drive space Database will take up: 211MB

#### Server – IIS Webserver

The recommended requirements on a server running Windows Server 2012 R2 and IIS are as follows:

* 1.4Ghz 64-bit CPU
* 1GB RAM
* 32GB Hard Drive Space
* Gigabit Ethernet (NIC)
* 1024 x 768 resolution or higher monitor

Unfortunately, I couldn’t find any minimum requirements for IIS itself as it is dependent on the load you expect it to take and the size of the files on the webserver. The maximum additional hard drive space that may be required due to web files can be seen below:

|  |  |  |
| --- | --- | --- |
| **Webserver** | | |
| **Type** | **Quantity** | **Space for 1 (Bytes)** |
| ASP.NET File (.aspx and .cs file) | 100 | 4,096 |
| PNG Image | 250 | 18,432 |

Total Size: 5017600 Bytes (4.8MB) + 10% overhead for OS = 5.3MB

So the webserver contents shall take up about 5.3MB on the server.

#### Server – Microsoft Exchange 2013 Mailserver

The recommended requirements for a server running Microsoft Exchange 2013 can be seen below:

* 64bit CPU
* 8GB RAM
* 30GB Hard Drive Space
* Gigabit Ethernet (NIC)
* 1024 x 768 resolution or higher monitor

These are the complete requirements for this server, as the hard drive space is a recommendation based on a normal mailserver load. Since my load will be significantly less than this, these requirements should be apt.

#### Client – Browser Load

Since the system will require users use a browser to access it, I got the minimum requirements for the modern browser Google Chrome, and they are displayed below:

* Windows Vista or later OS.
* Intel Pentium 4 CPU
* 350MB Disk Space
* 512MB RAM

These are the minimum requirements for a browser, however, it may just be easier to list a minimum requirement for the client system as being a modern browser and an internet connection.

### Requirements Specification

To Craig Sargent,

Below is a list of requirements pulled directly from our email communications and interviews. This lists all of the requirements of the system that is being produced, with a justification for each requirement.

|  |  |
| --- | --- |
| **Requirement** | **Justification** |
| Interface be web-based using ASP.NET server files creating HTTP-based web pages. | To allow any user to log-in and access the achievement system from any internet-connected device.  Question 3 in correspondence 1, interview 1 |
| Green and black colour-scheme | Requested by the user. Likely a reference to the hit sci-fi film The Matrix.  Question 4 in correspondence 1 & Question 2 in correspondence 2. |
| Leaderboard that compares user’s achievement scores, ranks and can filter by category. | Allows friendly competition between users.  Question 5 in correspondence 1 & Questions 4 & 6 in correspondence 2, interview 1. |
| Comprehensive administrator interface allowing the admin to: Create, delete and modify users, lock nicknames, create delete and modify achievements, achievement categories, ranks, rewards and the colour scheme. | To give the administrator complete control over the interface that the users are presented with.  Question 6 in correspondence 1 & Interview 1 |
| Achievement, rank and user data be stored on a MySQL database | To allow all of the data to be stored persistently and be accessible by a web client as well as easily be integrated into the existing school network. |
| The users be able to login using their school credentials, but still store data on the users like email preferences and nicknames. | To prevent confusion with usernames. |
| Achievements with names, descriptions, images and point-values. | Interview 1 & analysis of similar systems |
| Ability for users to request achievements be checked | To alert the administrator or teacher that the user feels they have fulfilled the criteria for the achievement.  Correspondence 2 question 1. |
| Ability for users to set a nickname, password and e-mail preferences. | Allows customisation of the user’s profile.  Interview 1 & correspondence 2 question 1 |
| Emails be sent when a user has obtained an achievement. | To alert the user when a response to their achievement request has been completed.  Correspondence 2 question 5 |
| A rewards system that tells the user what they have obtained upon reaching a certain rank. | To give the users more material reason besides from competition to achieve well, the system needs to display which rewards they get at each rank.  Interview 1 |
| Be compatible with a MySQL database, Microsoft Exchange 2013 and IIS Express. | For the system to operate and work in conjunction with existing systems.  Integration with existing systems. |
| The system will be pre-loaded with the previous system’s content, including achievements, ranks and rewards. | To allow the system to operate out-of-the-box  Interview 1 |

### Hardware & Software Requirements

Since the software has both a server element and a client element, both of these need to have separate requirements.

#### Server:

|  |  |
| --- | --- |
| **Hardware** | **Justification** |
| 1.4GHz 64-bit CPU | Minimum requirement for IIS. |
| 8GB RAM | Minimum requirement for Microsoft Express 2013. |
| Gigabit Ethernet | Minimum requirement for IIS & Microsoft Express. |
| VGA Monitor with a resolution 1024 x 768 or greater. | Minimum requirement for IIS & Microsoft Express. |
| **Software** | **Justification** |
| IIS Express webserver or similar  (32GB) | Required to display web elements. This is the webserver that all of the webpages that will be used for the interface will be pulled from. |
| MySQL Server V5.0+  (211MB) | Required to store persistent data. |
| Microsoft Exchange or similar  (30GB) | Required in order to send emails to the users. |

#### Client:

|  |  |
| --- | --- |
| **Hardware** | **Justification** |
| Keyboard | Required in order to log-in to and navigate the system. |
| Mouse | Required in order to navigate through on-screen menus |
| 500MB RAM | In order to store any assets the system is using and access them at a reasonable speed. |
| Internet Connection | To access the data and webpages. |
| VGA Monitor with a resolution 1024 x 768 or greater. | To display the contents of the webpages correctly. |
| **Software** | **Justification** |
| Modern Browser | Required so the webpages load efficiently and are displayed correctly. |

#### User review

If you are happy with these requirements, please could you sign the dotted line below, to bind these requirements as the agreement for the contract.

My Signature: …………………………………………………………….

Client’s Signature: …………………………………………………………….

# Design

## B. i) Nature of the solution

### Explanation of the Traditional SDLC Process

There are several different stages required to fully develop a solution, and without a proper plan of attack, the entire ordeal can become rather disorganised. There are several different standard approaches for planning and developing a full solution, which can be summarised as the Systems Development Life-Cycle (SDLC). As mentioned prior, there are several different ways to tackle the SDLC, and two of the ones that I am most likely to use are listed below:

#### Waterfall Model

The waterfall model allows a natural progression though the system’s development, without any need to go back and repeat stages, making it efficient as time is not wasted on stages that aren’t strictly necessary. Unfortunately, this can lead to poorer documentation if no time is spent going backwards and retracing after an issue in say, the design stage has arose while in implementation.

#### Spiral Model

The spiral model allows you to re-iterate the steps of the SDLC until the final work is completed. It is a form of RAD (Rapid application development) in which you return to previous steps to review what you have implemented and re-design the best approach going forward. Often, the design doesn’t change, and so you will go straight from evaluation back to implementation, but it is worth noting that the system offers you multiple chances for a redesign. I will be using this system to a certain extent as it allows great flexibility in the workflow. The primary issue with this solution is, however, that major redesigns can leave you project feeling broken up or new designs can simply just not work with previously created parts of the system, leaving you with no choice but to re-implement previous work. Another issue that this causes is a lack of understanding of the project from people, as the focus has shifted throughout the duration of the project.

Feasibility & Problem Definition

Maintenance

#### Hybrid Model

I have decided that my process will be a combination of both of these models. I shall primarily be using the waterfall model, but with occasional delves back into the spiral model for quick development, testing and presentation to the user to ensure that the application is on the right track and to allow additional requirements to be created before the solution is completed. This should allow my development to be reactive around the user, and not be strict when it comes to additional functionality being added.

### Systems Diagram

#### Web System

The entirety of the web-based system. Should allow both admins and users access to the achievement interface.

#### Web System / Login Screen

Provides login areas for both admins and users.

#### Web System / Login Screen / Admin Login

Allows Admins to login, will check user against their permission level to ensure only admins proceed to admin controls.

#### Web System / Login Screen / User Login

Allows users to login, if they do not have the required permission level to be an admin. Checks their username and password against the database. Password will be hashed upon creation using username as salt. When logging in, the same process will occur and the two results will be compared.

#### Web System / Login Screen / User Login / Reset Password

Allows users to reset their password by sending a password reset code or link to their registered e-mail address.

#### Web System / User Controls

Provides an interface for the user to use.

#### Web System / User Controls / Achievement Screen

Provides a list of the achievements and highlights the ones that the user has obtained.

#### Web System / User Controls / Leaderboard

Provides a leaderboard to the user, comparing their achievements and scores against other users’.

#### Web System / User Controls / Leaderboard / Filters

Gives a drop-down or similar menu that allows the user to configure filter options for the leaderboard.

#### Web System / User Controls / Leaderboard / Filters / Most Common Achievements

Allows the leaderboard to be filtered by the most common achievements in either ascending or descending order.

#### Web System / User Controls / Leaderboard / Filters / Best Students by points / rank

Allows the leaderboard to be filtered by student point count in either ascending or descending order.

#### Web System / User Controls / Leaderboard / Filters / Best Students by points by category

Allows the leaderboard to be filtered by student points, but splits it up into categories.

#### Web System / User Controls / Leaderboard / Filters / Set Categories

Allows the categories that are used for all of the filter options to be set.

#### Web System / User Controls / Overview

Provides a basic overview to the user including:

* The registered school username
* The nickname of the user, which changes colour if it is locked to red.
* The real name of the user
* The rank image for the user
* The name of the rank that the user has achieved
* The ranking of the rank that the user has achieved
* A progress bar showing the user’s progress to their next rank
* The information that the progress bar is displaying in both a fraction and percentile format
* The most recent achievements that the user has obtained.

#### Web System / User Settings

Provides an interface to the user to configure their user settings.

#### Web System / User Settings / Change Password

Allows the user to change the password associated with their account. This will re-perform the salting and hashing that occurs when the user is first created.

#### Web System / User Settings / Change E-mail Address

Allows the user to change the e-mail address associated with their account.

#### Web System / User Settings / Change E-mail Preferences

Allows the user to change their email preference to receive emails when something is changed. (e.g an achievement is awarded)

#### Web System / Admin Controls

Provides an interface to configure options about the web system

#### Web System / Admin Controls / User Administration

Provides an interface to configure and administrate users

#### Web System / Admin Controls / User Administration / Create User

Allows the creation of users. Adds a new row into the users table and gathers the data for this row.

#### Web System / Admin Controls / User Administration / Disable User

Allows users to be re-enabled after being disabled. Sets the disabled property in the users table to false.

#### Web System / Admin Controls / User Administration / Disable User

Allows the removal of users. Sets the disabled property in the users table to true.

#### Web System / Admin Controls / User Administration / Cleanup Users

Allows the removal of disabled, old users.

#### Web System / Admin Controls / User Administration / Elevate or Demote User

Allows a user to be elevated and granted admin privileges or be demoted and have these privileges revoked.

#### Web System / Admin Controls / User Administration / Modify User

Allows the modification of users properties.

#### Web System / Admin Controls / User Administration / Modify User / Reset Password

Changes the user’s password back to the system default.

#### Web System / Admin Controls / User Administration / Modify User / Set Nickname

Modifies the user’s nickname. Edits the Nickname attribute in the users table.

#### Web System / Admin Controls / User Administration / Modify User / Set Real name

Modifies the user’s set real name. Edits the realname attribute in the users table.

#### Web System / Admin Controls / User Administration / Modify User / Lock Nickname

Prevents further modification to the user’s nickname by the user. Sets the NickLocked property in the users table to true.

#### Web System / Admin Controls / User Administration / Modify User / Unlock Nickname

Allows modification to the user’s nickname by the user. Sets the NickLocked property in the users table to false.

#### Web System / Admin Controls / User Administration / Modify User / Reset Data

Removes all of the user’s achievement data. Removes all rows from the AchievementData and RewardData tables with the user’s ID and set’s the score attribute in the users table to zero.

#### Web System / Admin Controls / Achievement Approval

Allows the approval or disapproval of requests for achievements.

#### Web System / Admin Controls / Achievement Approval / Approve Request

Approves the request for the achievement. Changes the requested field to false.

#### Web System / Admin Controls / Achievement Approval / Deny Request

Denies the request for the achievement. Removes the associated row from the AchievementData Table.

#### Web System / Admin Controls / Distribute Achievements

Allows the admin to give specific achievements to specific users. Adds rows to the AchievementData Table with the associated UserID and AchievementID.

#### Web System / Admin Controls / Data Management

Provides an interface to allow the admin to change the data that is in the achievement system such as; the achievements, ranks, images and rewards.

#### Web System / Admin Controls / Data Management / Achievements

Allows the removal, addition and modification of achievements that the system has stored.

#### Web System / Admin Controls / Data Management / Achievements / Add

Adds an achievement. Name, Description, Category, Image, Value and whether or not it is hidden are fields to have data entered into. ID & DateAdded are auto-generated. Categories can either be selected from a list of existing categories that other achievements have set, or be set to something new. Adds a new row to the Achievements table.

#### Web System / Admin Controls / Data Management / Achievements / Remove

Removes an achievement. Removes the row from the achievement table.

#### Web System / Admin Controls / Data Management / Achievements / Modify

Allows the modification of achievements. Uses the same interface as adding an achievement but with all of the fields filled with the data that has already been set. Modifies the existing row in the Achievement table using the Unique ID.

#### Web System / Admin Controls / Data Management / Ranks

Provides an interface for the addition, removal and modification of ranks.

#### Web System / Admin Controls / Data Management / Ranks / Add

Allows the addition of ranks. Name, MinScore, and the image must be set by the user upon creation of the rank. Inserts a new row into the Ranks table.

#### Web System / Admin Controls / Data Management / Ranks / Remove

Allows the removal of ranks. Removes the row with the associated ID from the Ranks table.

#### Web System / Admin Controls / Data Management / Ranks / Modify

Allows the modification of ranks. Provides the same interface as adding the ranks, but with the fields filled out with the current data that is associated with that specific rank. Overwrites changes to the row with the same ID in the Ranks table.

#### Web System / Admin Controls / Data Management / Images

Provides and interface for addition and removal of images to the system. Allows images to be uploaded directly to the webserver, to be stored in a root folder.

#### Web System / Admin Controls / Data Management / Images / Add

Allows the uploading of images. Stores the images in an images folder in the root folder of the website. Allows other items, such as ranks and achievements to access these images’ URL’s for the image field in the database.

#### Web System / Admin Controls / Data Management / Images / Remove

Allows for the removal of images. Removes the images directly from the images folder in the root folder of the website.

#### Web System / Admin Controls / Data Management / Rewards

Provides and interface for the addition, removal and modification of rewards. These are purely cosmetic, not resulting in any action by the webpage, and merely serve as a marker for the admin to distribute physical rewards.

#### Web System / Admin Controls / Data Management / Rewards / Add

Allows the addition of rewards. Name, Image and descriptions are set by the admin upon creation of the reward. Creates a new row in the Rewards table, and auto-generates the ID.

#### Web System / Admin Controls / Data Management / Rewards / Remove

Allows the removal of rewards. Removes the associated row from the Rewards table.

#### Web System / Admin Controls / Data Management / Rewards / Modify

Allows for the modification of rewards. Provides the same interface as adding, but with the fields already containing the data of the reward that is being modified. Overwrites the existing row in the Rewards table with the associated ID.

### Directory Structure

### MySQL Database Structure

Part of the design requires that I design the full database structure before it is implemented. This will ensure that all variables have been accounted for and there are no redundant fields in the tables. Since MySQL has multiple types derived from parent types such as Integer and String, I have included an additional column with the type that the field will use in the database to ensure that it is completely clear what the type the field will contain. I have also added additional notes and information to the fields, so the database structure can be copied perfectly from the tables to the database without and further configuration than that stated.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Record Name** | **Type** | **Database Type** | **Default Value** | **Allow Null** | **Primary Key** | **Length (Bytes)** |
| **Achievements** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| Name | String | TinyText | "" | No | No | 255 |
| Description | String | TinyText | "" | No | No | 255 |
| Category | String | TinyText | "" | No | No | 255 |
| Image | String | TinyText | "" | No | No | 255 |
| Value | Integer | Int | 0 | No | No | 8 |
| DateAdded | DateTime | DateTime | DateTime Now | No | No | 8 |
| Hidden | Bool | Bit | 0 | No | No | 1 |
| **Rewards** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| Name | String | TinyText | "" | No | No | 255 |
| Description | String | TinyText | "" | No | No | 255 |
| Image | String | TinyText | "" | No | No | 255 |
| RankID | Integer | Int | 0 | No | No | 8 |
| **Users** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| Username | String | TinyText | "" | No | No | 255 |
| Password\* | String | String | "" | No | No | 65535 |
| RealName | String | TinyText | "" | No | No | 255 |
| NickName | String | TinyText | "" | No | No | 255 |
| Email | String | TinyText | "" | No | No | 255 |
| EmailPref | Boolean | Bit | 1 | No | No | 1 |
| DateAdded | DateTime | DateTime | "" | No | No | 8 |
| Score | Integer | Int | 0 | No | No | 8 |
| RankID | Integer | Int | 1 | No | No | 8 |
| NickLocked | Bool | Bit | 0 | No | No | 1 |
| Disabled | Bool | Bit | 0 | No | No | 1 |
| DateDisabled | DateTime | DateTime | NULL | Yes | No | 8 |
| PermissionLevel | Integer | Int | 0 | No | No | 8 |
| **AchievementData** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| UserID | Integer | Int | 0 | No | No | 8 |
| AchievementID | Integer | Int | 0 | No | No | 8 |
| Requested | Boolean | Bit | 0 | No | No | 1 |
| Awarded | Boolean | Bit | 0 | No | No | 1 |
| DateUpdated | DateTime | DateTime | DateTime Now (On Update) | No | No | 8 |
| **Ranks** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| Name | String | TinyText | "" | No | No | 255 |
| MinScore | Integer | Int | 0 | No | No | 8 |
| Image | String | TinyText | "" | No | No | 255 |
| **Rewards Data** | | | | | | |
| ID | Integer | Int | Auto-Increment | No | Yes | 8 |
| UserID | Integer | Int | "" | No | No | 8 |
| RewardID | Integer | Int | "" | No | No | 8 |
| Awarded | Boolean | Bit | 0 | No | No | 1 |

#### Notes

|  |  |
| --- | --- |
| **Record Name** | **Note** |
| **Achievements** | |
| ID | The unique identifier for the achievement |
| Name | The name of the achievement |
| Description | The description of the achievement |
| Category | The category of the achievement |
| Image | The URL for the image associated with the achievement |
| Value | The value of the achievement in points |
| DateAdded | The date that the achievement was added |
| Hidden | Whether or not the achievement is hidden until achieved. 0 = No |
| **Rewards** | |
| ID | The unique identifier for the reward |
| Name | The name of the reward |
| Description | The description of the reward |
| Image | The URL for the image associated with the reward |
| RankID | The ID of the rank that is required in order to obtain the reward |
| **Users** | |
| ID | The unique identifier for the user |
| Username | The username for the user |
| Password | The password for the user, salted and hashed using PBKDF2\*\* |
| RealName | The real name of the user, used for identification purposes |
| NickName | The leaderboard nickname for the user |
| Email | The email address of the user |
| EmailPref | The email preference of the user. 0 = No email updates |
| DateAdded | The date that the user was added |
| Score | The total score of the user |
| RankID | The ID of the associated rank that the user has achieved |
| NickLocked | Whether or not the user's nickname has been locked. 0 = No |
| Disabled | Whether or not the user has been disabled. 0 = No. |
| DateDisabled | The date the user was disabled at |
| PermissionLevel | The permission level of the user, used to set administrators. 0 = User |
| **AchievementData** | |
| ID | The unique identifier for the achievement data |
| UserID | The ID of the associated user that the achievement has been awarded to |
| AchievementID | The ID of the associated achievement that has been awarded to the user |
| Requested | Whether or not the user has requested this achievement. 0 = No. If no, assumed that the user now has this achievement |
| Awarded | The awarded status of this achievement. If 0, user will be presented with achievement upon next login, and set to 1. |
| DateUpdated | The date of the last update to this row. |
| **Ranks** | |
| ID | The unique identifier for the rank |
| Name | The name of the rank |
| MinScore | The minimum score required in order to achieve the rank |
| Image | The URL for the image associated with the rank |
| **Rewards Data** | |
| ID | The unique identifier for the reward data |
| UserID | The ID of the associated user that the reward has been given to. |
| RewardID | The ID of the reward that has been given to the user. |
| Awarded | The awarded status of this reward. If 0, user will be presented with this reward upon next login, and set to 1. |

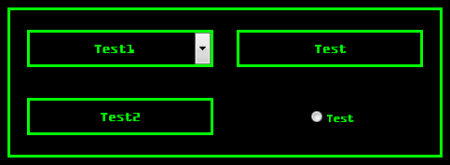
\* The password column should be named anything but password. I’m actually going to name it “sahp” – Salt and hashed password.

\*\*This description should be omitted from the actual table and is only here for completion.

### Overarching Interface Designs

The following parts of the interface will be overarching and shall be consistent for both users and administrators.

#### CSS Style

Since the entirety of the system will be using ASP.NET, Cascading Style Sheets (CSS) are available to make configurable colour schemes, and while a default colour scheme will come pre-set with the system, it will also be configurable, allowing the user to change the system’s style at a whim. The following CSS stylesheet is not exhaustive, and may need to be added to, but gives a good feel for what style the system will have at the end. All of the controls in the following layouts will adhere to the style set out in the CSS stylesheet.

.base

{

background-color:black;

color:lime;

text-align: center;

font-size: 12px;

font-family: 'OCR A';

font-weight: bold;

}

.base\_w\_border

{

background-color:black;

color:lime;

text-align: center;

font-size: 12px;

font-family: 'OCR A';

font-weight: bold;

border:3px solid lime;

}

.table

{

border:3px solid lime;

border-collapse: separate;

background-color:black;

padding: 4px;

color:lime;

text-align: center;

font-size: 12px;

font-family: 'OCR A';

font-weight: bold;

width: 100%;

height: 100%;

}

.header

{

border:3px solid lime;

background-color:black;

color:lime;

text-align: center;

font-size: 28px;

font-family: 'OCR A';

font-weight: bold;

}

.button

{

border:3px solid lime;

cursor:pointer;

background-color:black;

color:lime;

width: 200px;

height: 40px;

text-align: center;

font-size: 14px;

font-family: 'OCR A';

font-weight: bold;

}

.button:hover

{

border:2px solid lime;

}

.DropDown

{

border:3px solid lime;

cursor:pointer;

background-color:black;

color:lime;

width: 200px;

height: 40px;

text-align: center;

font-size: 14px;

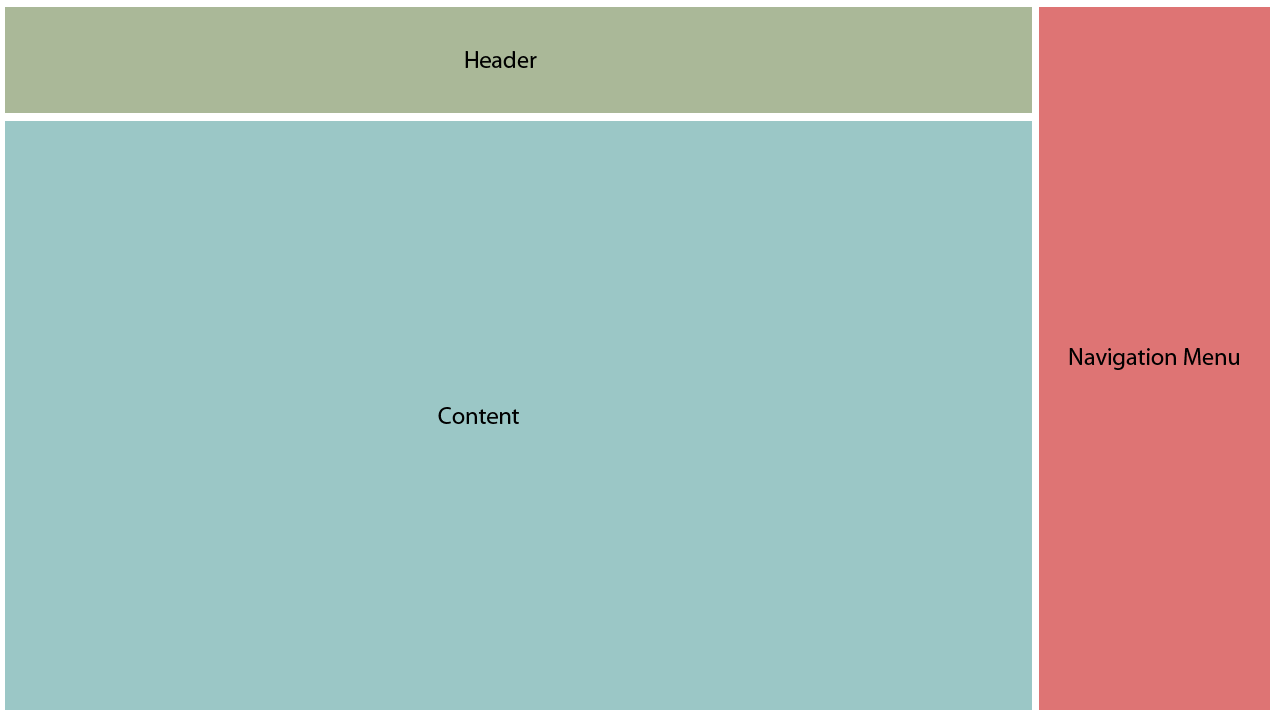
font-family: 'OCR A';

font-weight: bold;

}

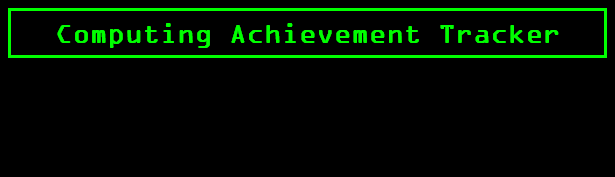
#### Master Page Layout

The master page, or parent page, will have the following layout. Each box in this layout will be a placeholder control, to be overwritten by other controls and pages, dynamically.



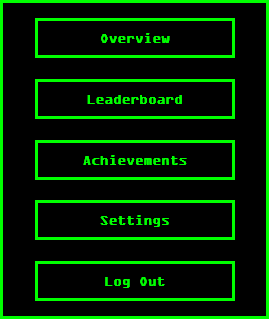
#### Header

The header will just be a configurable text box. The contents and style of which can be changed. By default, it will look like this:



#### Navigation Menu

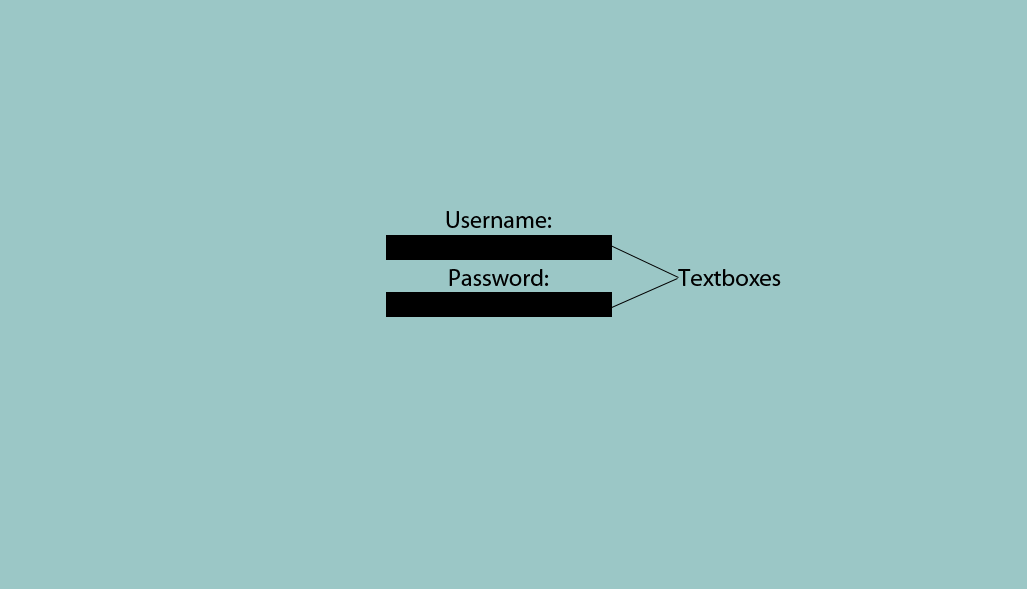
The navigation menu will be a list of buttons in a single column table that link to the various pages the user has access to. By default, it will look something like this:



It is worth noting that the admin will have a different variant, as they will have access to different pages.

#### Logon Screen

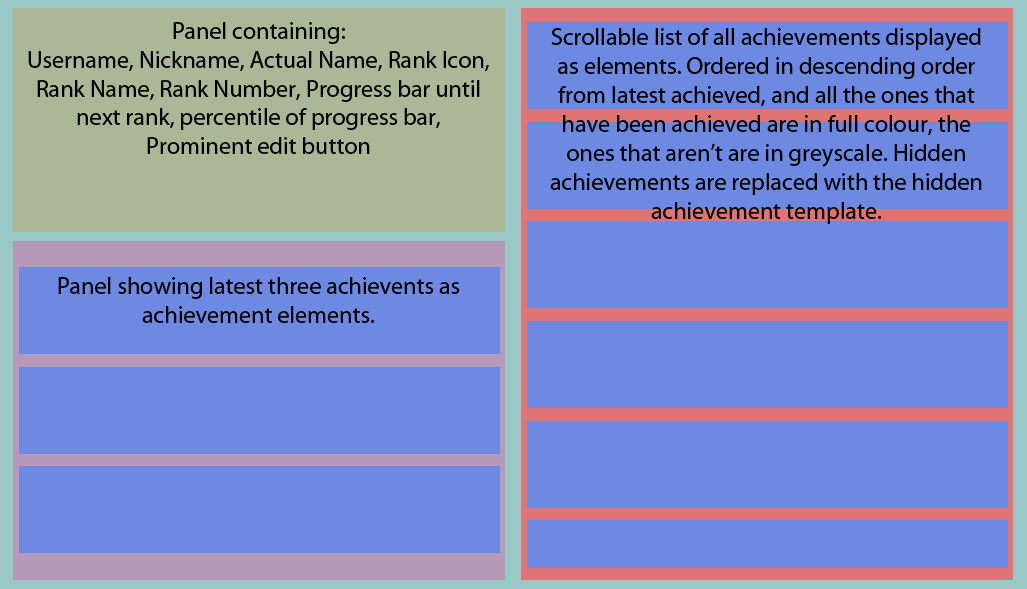
The logon screen is used to identify which user to load, by allowing the user to log in to the system. This is also effectively the home page, as the project will be integrated via link into an existing website.



### User Interface Designs

#### Overview

The overview page is effectively the user home page, and will provide all important information to the user at once.



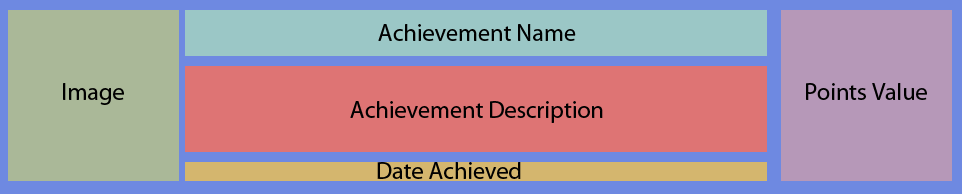
#### Overview Panel

The above overview page shows an overview panel, which will display most information about the user. The contents of this panel is as follows:



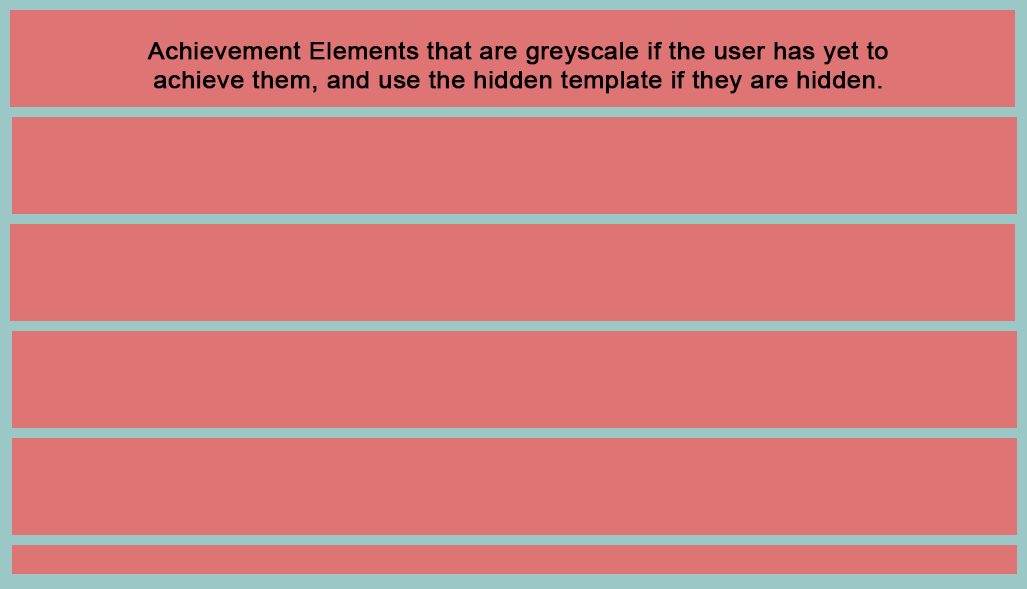
#### Achievement Element

The overview also displays the use of an achievement element. This is a control that is bound to a single achievement, and displays all of the information relating to that achievement. Its layout will be as follows:



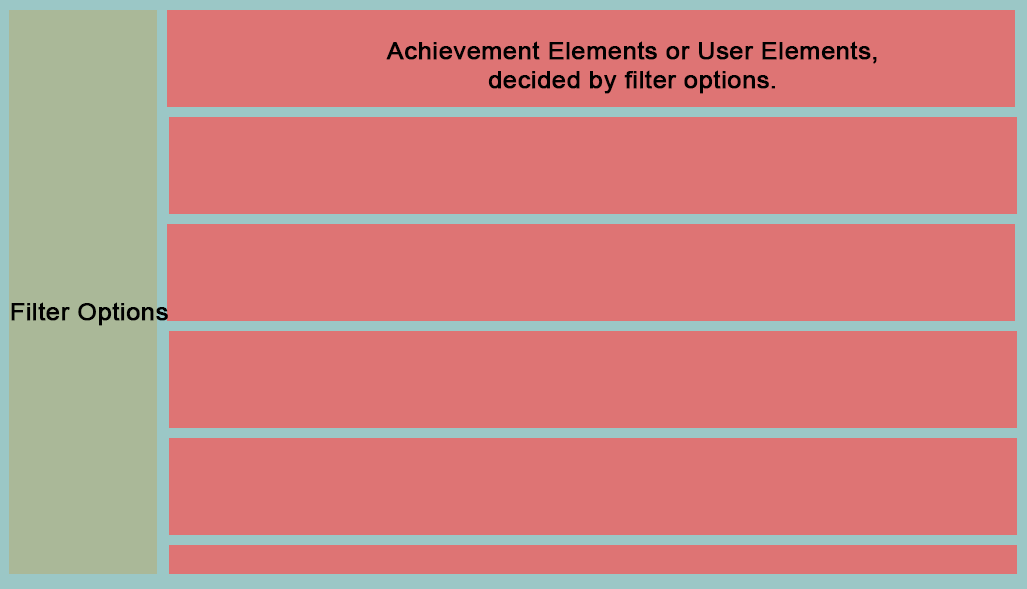
#### Achievement Screen

The achievement screen will display all achievements in a nice big space with no extra stuff to get in the way. Achievements will be organised by date achieved (if at all) with achievements not already achieved below those that have. They will also be organised into the various categories. Each achievement element that has not been achieved will have a button overlayed, which allows the user to request upto 5 achievements.



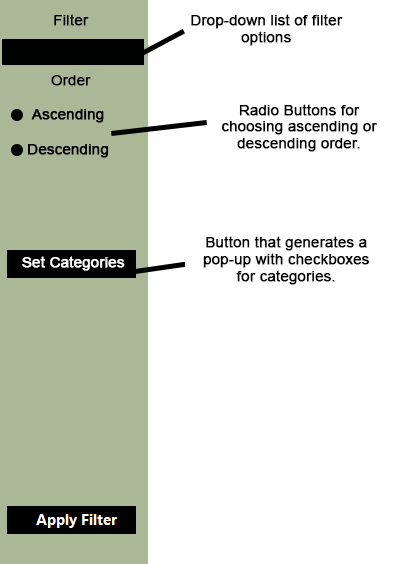
#### Leaderboard

The leaderboard is used to compare statistics from other users, and has various filter options to change which achievements or users to display. It will follow this layout:



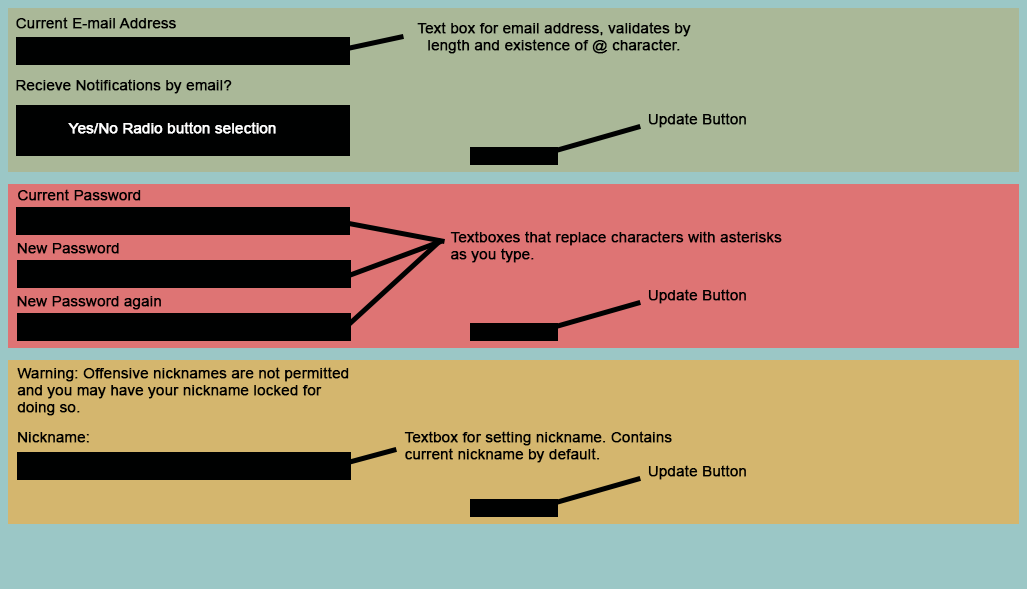
#### Filter Options

The filter options of the previous page, the leaderboard, will contain several controls for changing the contents of the leaderboard.



#### User Settings

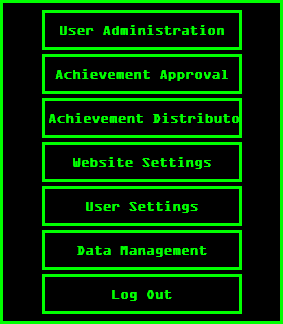
The user settings page will allow the user to update their email address and email preference, change their password and set their nickname. All of the text boxes will be validated for length between 4-64 characters, the e-mail address will be validated for existence of the ‘@’ symbol, the password will be validated for existence of a symbol/number as well as a letter, and the nickname will be locked to numbers and letters only. Each section has its own update button and corresponds to an update of the fields of that section only. Here is the layout:



### Admin Interface Designs

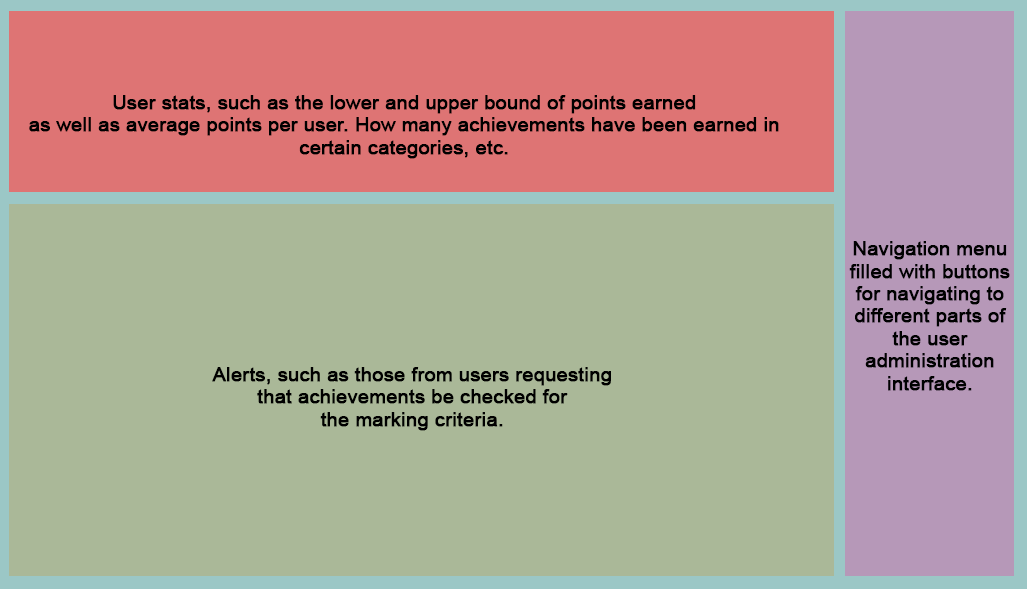
#### Navigation Menu

As mentioned before, the navigation menu will change its contents based on what the user has access to. The admin navigation menu will look something like this:



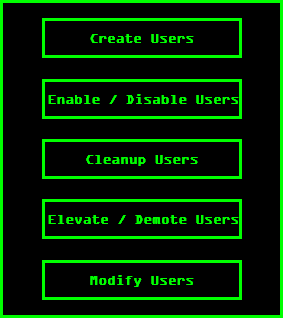
#### User Administration Overview

This screen will contain user stats such as lower bound, upper bound and average points for users, how many achievement requests there are, and a navigation menu for all of the separate parts of user administration section of the system.



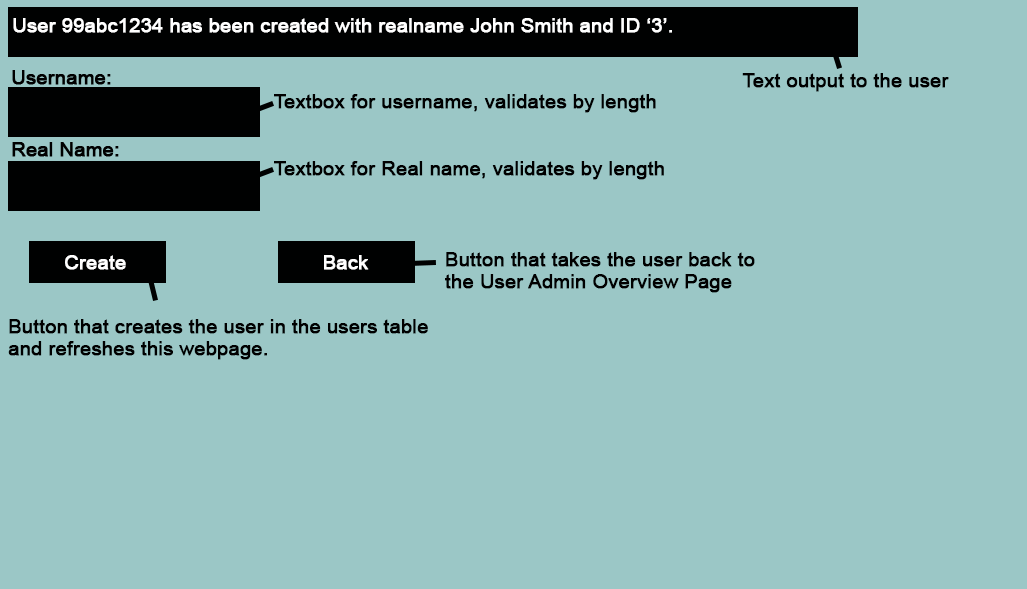
#### User Administration Overview Navigation menu

The navigation menu from the previous design will likely look like this:



#### Create Users

This page will have two simple textboxes in which the user will enter the desired user’s username and real name, and the user will be added to the users table. Information for each record that is not entered by this form is created automatically, either by the algorithm that the create button runs, or by the database itself. The text output will let the user know if the last operation was successful. The back button returns the user to the User Administration Overview page.



#### Enable / Disable Users

The Enable / Disable users page will handle the enabled state of users. The page will mostly comprise of a datagrid, showing the user’s username, nickname, realname and a button on each row. The button will have contextual text and style, and will be directly linked to that user’s disabled property in the users table. When this property is true, the button will show disabled, when this property is false, the button will show enabled. When disabled, the final column will show the date that the user was disabled. More information columns for each user may be added, along with filter functionality if there is time.

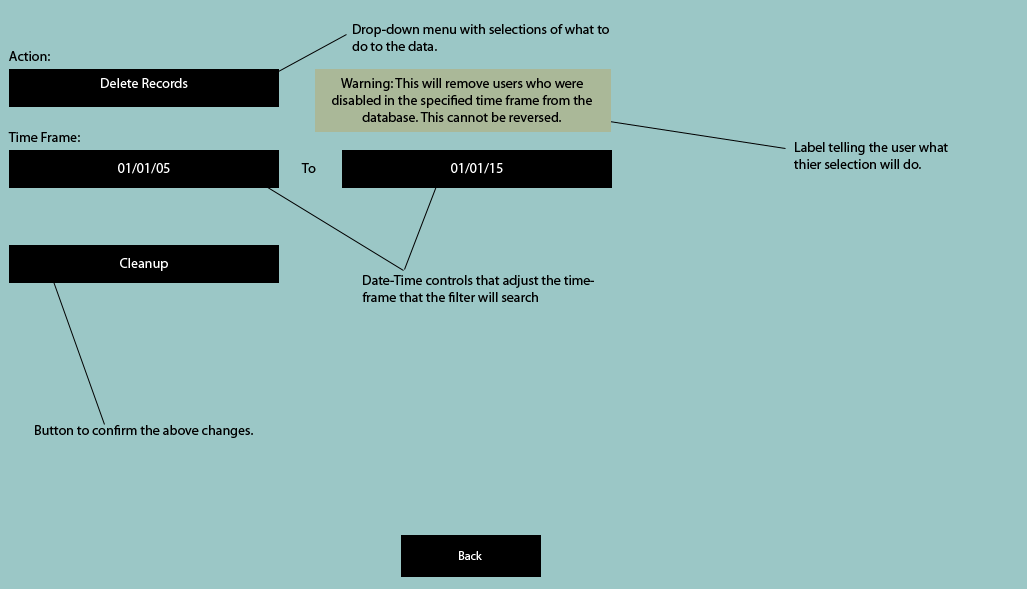


#### Cleanup Users

This page allows the admin to remove users that have been disabled, in order to clean up the users table. The first drop-down box control will allow the admin to choose what will happen to the data, which will change the text-box to the right of the control, and will tell them what their selection will actually do. The current selections currently are:

* Remove the users from the database entirely.
* Move the users to a separate table

The time frame will adjust the dates that the cleanup process will look for when cleaning users. It will only clean users that were disabled in that time frame.



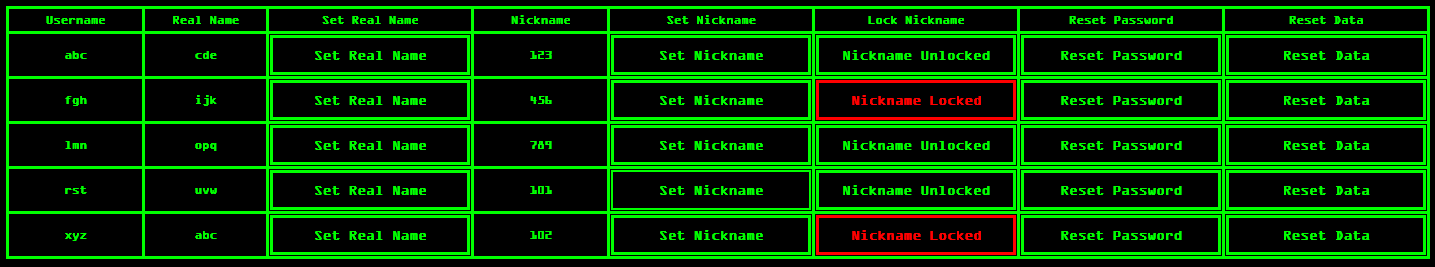
#### Elevate / Demote Users

Similar to the enable / disable users page, will contain a datagrid of users with buttons associated to each one. Pressing the button will change their permission property in the users table to 0 - User and 255 - Admin respectively.



#### Modify Users

With the modify users page, things get a little complicated. Since this screen will be handling the modification of 5 different properties of a user, it needed to be quite comprehensive, and it warranted prototyping. This is what I have come up with:



An eight-columned datagrid. Each row signifies a user. The columns are (in order):

* Username
  + The user’s username – pulled from the users table.
* Real Name
  + The user’s real name - pulled from the users table.
* Set Real Name
  + Button that, upon clicking, displays a pop-up window allowing the admin to change the user’s name.
* Nickname
  + The user’s nickname – pulled from the users table.
* Set Nickname
  + A button that, upon clicking, displays a pop-up window that allows the admin to change the user’s nickname.
* Lock Nickname
  + A toggle-able button that changes the style to the disabled style when clicked off. This locks the user’s nickname, preventing the user from changing it.
* Reset Password
  + A button that, upon pressing, opens up a dialog, confirming the admin’s choice. If yes is then clicked, the user’s password is reset to the default.
* Reset Data
  + A button that, upon pressing, removes all data associated with that user, providing a clean slate for the user.

#### Achievement Approval

This page will allow the admin to approve or deny achievements that have been requested by users. Each row will signify a single achievement request from a single user. The list will be ordered alphabetically by user name and then achievement name. Each row will have an approve and a deny button, allowing each request to be approved and denied, individually.



#### Distribute Achievements

This page allows the administrator to both add achievements to users, and revoke them. By selecting rows in the green datagrid, and pressing the remove selected button, the selected achievements will be removed from the selected users after a pop-up confirming their selection confirms the change. To give an achievement to a user, the user is selected from a drop down box in the purple section, which displays the user’s username, realname and nickname, all with slashes in between. The admin then selects the achievement from the next drop down box, where achievements already given are greyed out and unselectable. Finally, pressing the add achievement button will give the selected user the selected achievement.

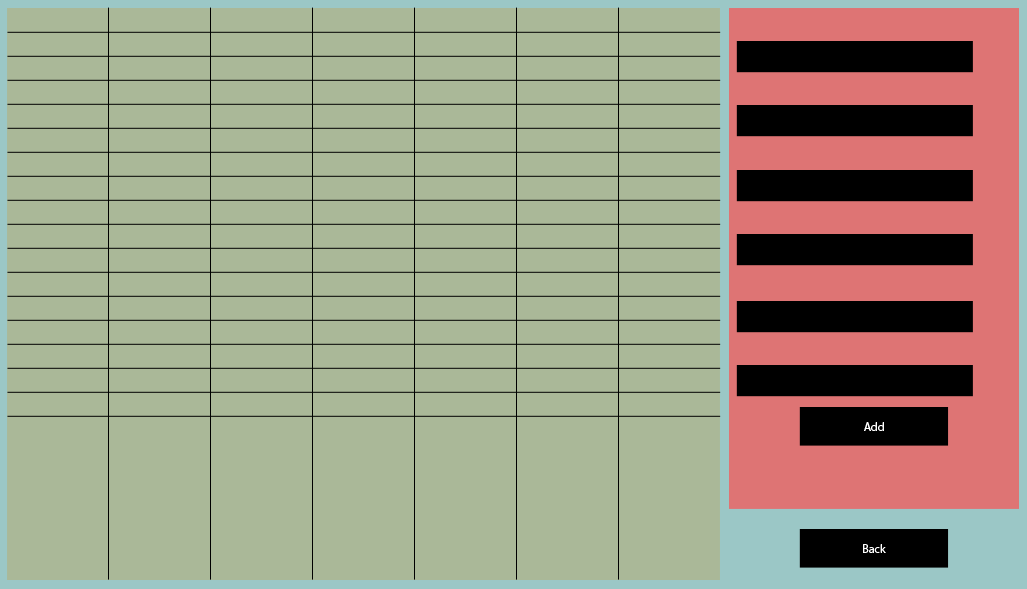


#### Data Management

The data management page is a bridging page that provides links or buttons to the achievement manager, rank manager, image manager and reward manager.

#### Achievement, Rank & Reward Management Interface

Since the data in these records is going to entirely be text, it makes sense that their interface will be very similar. Here’s the template for what I think that these interfaces should look like:

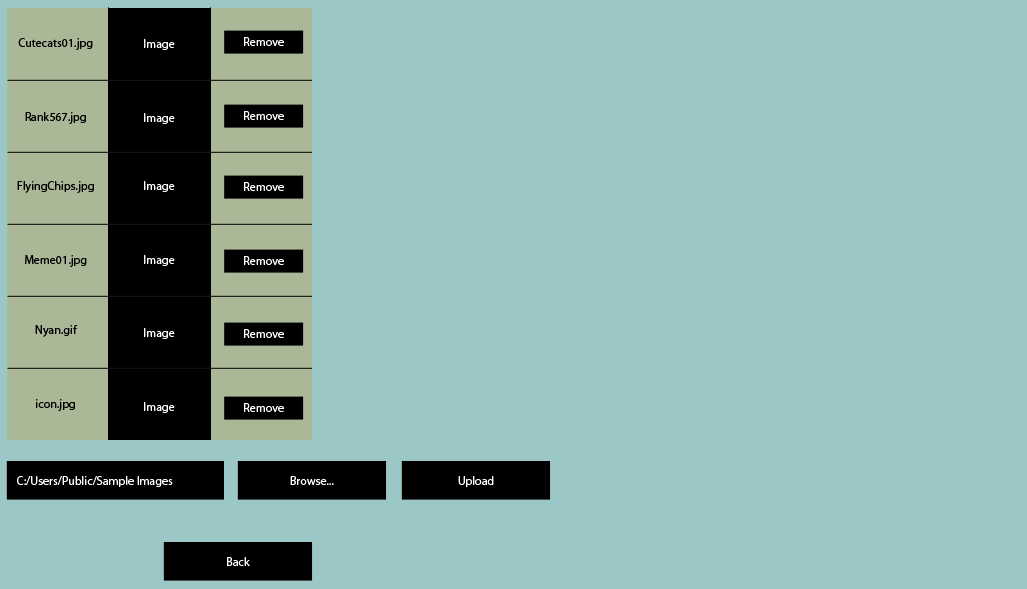


Now, as you can see, this interface diagram is very barebones in comparison to the others, so allow me to elaborate. The green data grid is where all of the currently data in the system is represented. Each record in the table in the database is represented by a row, and will contain most of the data from these rows (Information on which columns to include below). Each row will also have a remove button at the end for removing that specific row. The red interface on the side allows you to add and remove data. The majority of this interface is used by fields used for data entry. Some of these will be textboxes, some of them will be drop-down lists. In addition, there is the usual back button to take you to the previous screen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Interface** | Achievement | Rank | Reward |
| **Field / Column 1** | Textbox / Name | Textbox / Name | Textbox / Name |
| **Field / Column 2** | Textbox / Description | Textbox / Minimum Score | Textbox / Description |
| **Field / Column 3** | Textbox / Category | Drop-down / Image | Drop-down / Image |
| **Field / Column 4** | Drop-down / Image |  |  |
| **Field / Column 5** | Textbox / Value |  |  |
| **Field / Column 6** | Drop-down / Hidden |  |  |

#### Image Management

As multiple of the previous elements require images to be associated with them (Achievements, awards, ranks), there needs to be a way of uploading and removing images to and from the server with ease. This is where the image management screen comes in. The green element contains all of the images, with their names and a remove button in a datagrid. Each image is represented by a row. Below this, or perhaps to the side, there shall be a textbox, used for defining an image path, alongside a browse button, to open up the file browser interface native to the OS, and an upload button, to upload the selected file to the webserver. All images uploaded via this will be stored in the location root/Images.



### User Sign off Design Proposal

If you are happy with these designs and agree that this is what the system entail, please could you sign the dotted line below, to bind these designs as the agreement for the contract.

My Signature: …………………………………………………………….

Client’s Signature: …………………………………………………………….

## B. ii) Algorithms

### Database

#### GetAchievements(List of String categories)

1. Create the query using the list of categories. Get data from achievement table, all records with any of the strings as the ‘Category’ column.
2. Open the connection if it isn’t alreadyser
3. Execute the query.
4. Open the datareader and get the return.
5. Place the return into a list of achievements.
6. Close the datareader.
7. Return the new list of achievements.

#### GetAchievementCounts(List of String categories)

1. Take the categories, use GetAchievements to get a list of achievements that are in those categories.
2. Convert the achievement list to a dictionary: Dictionary<Achievement, integer>
3. Foreach of the achievements in the dictionary:
   1. Set the query to get the number of that achievement from the achievementData table.
   2. Open the connection if it isn’t already
   3. Execute the query
   4. Open the datareader and get the return
   5. Set the value for that achievement in the dictionary to the integer returned.
   6. Close the datareader.
4. Return the new dictionary.

#### GetUserAchievements(User user)

1. Set the query to get the all the achievementIDs from achievementdata table with the userID of this user.
2. Open the connection if it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Save all of the return as a list of integers.
6. Close the data reader.
7. For each integer in the previous list of integers
   1. Set the query to obtain that achievement from the achievement table.
   2. Execute the query
   3. Open the data reader and get the return
   4. Add the data to an achievement object and add it to a list of achievements
   5. Close the data reader
8. Set the integer list to be null.
9. Return the achievement list.

#### GetUserScoresForCategories(List of String categories)

1. Make a double Dictionary<String (Primary Key), Dictionary<User (Secondary Key), Integer>>
2. Place the category list into the key for the primary dictionary.
3. Use GetUsers to get a list of users.
4. Place the users list into the secondary key of each primary key.
5. For each category
   1. For each user
      1. Set the query to get the achievements ID’s that the user has obtained.
      2. Open the connection if it isn’t already
      3. Execute the query
      4. Open the data reader and get the return.
      5. Create a list of ID’s from the return.
      6. Close the data reader
      7. Get the total score that the achievements are worth using GetAchievementsScore
      8. Set the Integer in the original double dictionary for this user to that value.
6. Return the double dictionary.

#### GetAchievementsScore(List of integers IDs, List of strings Categories)

1. Take the categories, use GetAchievements to get a list of achievements that are in those categories.
2. For each of the integers in the IDs list
   1. For each of the achievements in the achievement list
      1. If the integer and the ID of the achievement match,
         1. Add the value of the achievement to the score total.
3. Return the score total

#### GetUsers()

1. Create a query to get all users from the table users.
2. Open the connection if it isn’t already
3. Execute the query.
4. Open the data reader and get the return
5. Place the return into a list of Users
6. Close the data reader
7. Return the users.

#### ExecuteNonQuery(string query)

1. Take the query
2. Open the connection if it isn’t already
3. Execute the query
4. Return if query was successfully executed

### Leaderboard

#### addAchievementsToGrid(List of Achievement achievements)

1. Set columns to achievement type.
2. For each Achievement in achievements
   1. Add row to data grid, set data to that of achievement

#### addUsersToGrid(List of User users)

1. Set columns to user type.
2. For each User in users
   1. Add row to data grid, set data to that of achievement

#### sortAchievementsAscending(Dictionary<Achievement, Integer> dictionary)

1. Using LINQ, sort the dictionary in ascending order by value.
2. Convert LINQ IOrderedEnumerable<Achievement, Integer> to Dictionary<Achievement, Integer> and return.

#### sortAchievementsDescending()

1. Using LINQ, sort the dictionary in descending order by value.
2. Convert LINQ IOrderedEnumerable<Achievement, Integer> to Dictionary<Achievement, Integer> and return.

#### sortStudentPointsAscending()

1. Using LINQ, sort the dictionary in ascending order by value.
2. Convert LINQ IOrderedEnumerable<User, Integer> to Dictionary<User, Integer> and return.

#### sortStudentPointsDescending()

1. Using LINQ, sort the dictionary in descending order by value.
2. Convert LINQ IOrderedEnumerable<User, Integer> to Dictionary<User, Integer> and return.

#### filterSort()

1. Pull the filter string from the drop-down list text value.
2. Pull the ascending/descending order boolean from the radio buttons.
3. Pull a list of categories from the selected categories.
4. Switch statement for filter:
5. If “Most common”
   1. Pull achievement counts dictionary Dictionary<Achievement,Integer> using database.GetAchievementCounts using the categories pulled earlier.
   2. If order is true
      1. Perform sortAchievementsAscending using the above dictionary and get the return
   3. Else
      1. Perform sortAchievementsDescending using the above dictionary and get the return
   4. Foreach achievement in the above return,
      1. Add it to a list of achievements.
6. If “Best student”
   1. Pull the studentPointCounts dictionary Dictionary<User,Integer> using database.GetUserScoresForCategories using the above pulled categories.
   2. If order is true
      1. Perform sortStudentPointsAscending using the above dictionary and get the return
   3. Else
      1. Perform sortStudentPointsDescending using the above dictionary and get the return
   4. Foreach User in the above return,
      1. Add it to a list of Users.
7. Depending on which switch was performed, add either the users or the achievements to the table.

## iii) Test Strategy

[Insert things here]

### Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | What is being tested | Input data or actions | Expected outcome |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# C. Software Development and Testing

## Initial class setup and database queries

### 27th November 2015

#### Achievement Type (App\_Code\Objects\Achievement.cs)

Work was done to create the achievement type. These variables were defined:

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Type** | **Identifier** |
| Public, Read-Only | Int | ID |
| Public, Read-Only | String | Name |
| Public, Read-Only | String | Description |
| Public, Read-Only | String | Category |
| Public, Read-Only | String | Image |
| Public, Read-Only | Int | Value |
| Public, Read-Only | DateTime | DateAdded |
| Public, Read-Only | Bool | Hidden |

In order to make the variables read only, a private instance of them had to be setup, with the public instance set to read the value from the private instance. This means that only procedures within the class can set the variables. Like so:

private int id;

public int ID { get { return id; } }

Two constructors were added. One is used to create a new record in the database, one is used for existing records. The constructor for adding to the database is not complete. The code is below:

/// <summary>

/// Creates an entirely new achievement class and adds it to the database.

/// Then gets database-specific properties and pulls them too.

/// </summary>

public Achievement(string name, string description, string category,

string image, int value, bool hidden)

{

// TODO: Database.AddAchievement(name, description, category, image,

// value, hidden);

}

/// <summary>

/// Constructor used by the database class to create achievements returned

/// from the achievements table

/// </summary>

public Achievement(int id, string name, string description, string category,

string image, int value, DateTime dateAdded, bool hidden)

{

this.id = id;

this.name = name;

this.description = description;

this.category = category;

this.image = image;

this.value = value;

this.dateAdded = dateAdded;

this.hidden = hidden;

}

#### Rank Type (App\_Code\Objects\Rank.cs)

Work was completed to create the Rank type. These variables were defined. Read only was achieved in the same way it was for the Achievement type.

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Type** | **Identifier** |
| Public, Read-Only | Int | ID |
| Public, Read-Only | String | Name |
| Public, Read-Only | Int | MinScore |
| Public, Read-Only | String | Image |

Constructors were made to operate in the same way as the achievement class. Here is the code:

/// <summary>

/// Creates an entirely new rank class and then adds it to the rank table. Then gets

/// database-specific properties and pulls them too.

/// </summary>

public Rank(string name, int minScore, string image)

{

// TODO: this = Database.CreateRank(name, minScore, image);

}

/// <summary>

/// Consructor used by the database class to create ranks returned from the ranks table.

/// </summary>

public Rank(int id, string name, int minScore, string image)

{

this.id = id;

this.name = name;

this.minScore = minScore;

this.image = image;

}

#### Reward Type (App\_Code\Objects\Reward.cs)

Again, work was done to create the reward type. These are the variables that were defined. Read-only was achieved in the same way once again, and will be presumed unless otherwise specified from this point forward.

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Type** | **Identifier** |
| Public, Read-Only | Int | ID |
| Public, Read-Only | String | Name |
| Public, Read-Only | String | Description |
| Public, Read-Only | String | Image |
| Public, Read-Only | Int | RankID |

Constructors were made to operate in the same way again. Here is the code:

/// <summary>

/// Creates an entirely new reward class and then adds it to the reward table. Then gets

/// database-specific properties and pulls them too.

/// </summary>

public Reward(string name, string description, string image, int rankID)

{

// TODO: Database.CreateReward(name, description, image, rankID);

}

/// <summary>

/// Consructor used by the database class to create rewards returned from the rewards table.

/// </summary>

public Reward(int id, string name, string description, string image, int rankID)

{

this.id = id;

this.name = name;

this.description = description;

this.image = image;

this.rankID = rankID;

}

#### User Type (App\_Code\Objects\User.cs)

The User type was created and the following variables were defined:

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Type** | **Identifier** |
| Public, Read-Only | Int | ID |
| Public, Read-Only | String | Username |
| Public, Read-Only | DateTime | DateAdded |
| Public, Read-Only | Int | Score |
| Public, Read-Only | Int | RankID |
| Public, Read-Only | Bool | Disabled |
| Public, Read-Only | DateTime | DateDisabled |
| Public | String | RealName |
| Public | String | NickName |
| Public | String | EmailAddress |
| Public | Bool | EmailPreference |
| Public | Bool | NickLocked |
| Public | Int | PermissionLevel |

The constructors were made to operate in the same way. Here is the code:

/// <summary>

/// Create completely new user. Adds this user to the database. Then gets

/// database-specific properties and pulls them too.

/// </summary>

public User(int id, string username, string password)

{

// TODO: this = Database.CreateUser(id, username, password)

}

/// <summary>

/// Consructor used by the database class to create users returned from the users table.

/// </summary>

public User(int id, string username, string realName, string nickName, string email, bool emailPref, DateTime dateAdded, int score, bool nickLocked, bool disabled, int permissionLevel)

{

this.id = id;

this.username = username;

this.RealName = realName;

this.NickName = nickName;

this.EmailAddress = email;

this.EmailPreference = emailPref;

this.dateAdded = dateAdded;

this.score = score;

this.NickLocked = nickLocked;

this.disabled = disabled;

this.PermissionLevel = permissionLevel;

}

#### Database Class (App\_Code\Database.cs)

The database class was created. This is where all interaction with the MySQL database will take place. It is a MySQL Controller, if you will. The following variables were defined:

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Type** | **Identifier** |
| Private | OdbcConnection | connection |
| Private Const | String | userTable |
| Private Const | String | achievementTable |
| Private Const | String | achievementDataTable |
| Private Const | String | rewardsTable |
| Private Const | String | rewardsDataTable |
| Private | String | error |

The connection variable is immediately defined from the Web.config file, the constants are defined with default values and the error variable is made to display output to a console window if it is changed. The code for those is below:

private OdbcConnection connection = new OdbcConnection(ConfigurationManager.ConnectionStrings["MySQLConnStr"].ConnectionString);

// Bunch of constants that can be used if there is a change in the database's table names.

private const string userTable = "users";

private const string achievementTable = "achievements";

private const string achievementDataTable = "achievementdata";

private const string rewardsTable = "rewards";

private const string rewardsDataTable = "rewardstable";

/// <summary>

/// Variable that when set, displays output to console window. Alerts the admin of issues.

/// </summary>

private string error

{

set

{

Console.WriteLine(value);

}

}

Connection procedures were created. OpenConnection, CloseConnection, and connectionOpen were created. OpenConnection opens the connection to the database. CloseConnection closes the connection to the database, and connectionOpen checks if the connection is open, and if it is not, opens it. The code for these is on the next page.

/// <summary>

/// Opens the connection and allows queries to be executed. Returns true if successful.

/// </summary>

public bool OpenConnection()

{

try

{

connection.Open();

return true;

}

catch (OdbcException ex)

{

switch (ex.ErrorCode)

{

case 0:

error = "Failed to connect to the database.";

break;

case 1045:

error = "Invalid username/password. Please try again.";

break;

case 1042:

error = "Unable to create the socket on port 3306. Please check your network configuration.";

break;

default:

error = "Error " + ex.ErrorCode + " has occured. Please report this

issue.";

break;

}

return false;

}

}

/// <summary>

/// Closes the connection. Will keep trying until it manages to do so. Should only occur open the program closing.

/// </summary>

public void CloseConnection()

{

do

{

try

{

connection.Close();

}

catch (OdbcException ex)

{

error = ex.Message;

}

}

while (connection.State != System.Data.ConnectionState.Closed);

}

/// <summary>

/// Check to see if the connection is open. If it is not, tries to open the connection. If this fails, returns false. If connection is open or is successfully opened, returns true.

/// </summary>

private bool connectionOpen()

{

if (connection.State == System.Data.ConnectionState.Open)

{

return true;

}

else

{

return OpenConnection();

}

}

Some of the query functions were created. GetAchievements, GetAchievementCounts and ExecuteNonQuery were coded. The first two were defined in the algorithms stage, so I will not go into detail about what they do here, but ExecuteNonQuery is for queries that do not require a return. The code for these functions is below:

/// <summary>

/// Executes a query without a return.

/// </summary>

private bool ExecuteNonQuery(string query)

{

if (connectionOpen())

{

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

try

{

cmd.ExecuteNonQueryAsync(); // Executes the command

return true;

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

return false;

}

}

else

{

return false;

}

}

/// <summary>

/// Returns a list of achievements that are of the specified categories.

/// </summary>

public List<Achievement> GetAchievements(List<string> categories)

{

// Create a list to store the returned achievements

List<Achievement> achievements = new List<Achievement>();

// Get all achievements where the category is...

string query = "SELECT \* FROM `" + achievementTable + "` WHERE `Category` IN (";

// ..any of the categories specified.

foreach (string x in categories)

{

query = (query + '"' + x + '"' + ",");

}

// Remove the succeeding comma that is added due to the previous foreach loop,

// and finish the query with a bracket.

query = query.Remove(query.Length - 1);

query = query + ")";

if (connectionOpen())

{

// Create a database command from the query and existing connection.

OdbcCommand cmd = new OdbcCommand(query, connection);

try

{

// Execute the command and open a reader.

OdbcDataReader dataReader = cmd.ExecuteReader();

while (dataReader.Read()) ; // Read the next record.

{

// Get the id first, and check to make sure it is something.

int id = dataReader.GetInt16(0);

if (id != 0)

{

// Adds the achievement that is read from the database to the

// list of achievements that is to be returned.

achievements.Add(new Achievement(id, dataReader.GetString(1),

dataReader.GetString(2), dataReader.GetString(3),

dataReader.GetString(4), dataReader.GetInt16(5),

dataReader.GetDateTime(6), dataReader.GetBoolean(7)));

}

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

return achievements;

}

/// <summary>

/// Returns a dictionary of achievements and the number of times they have been achieved,

/// based on the categories set from the achievementData table.

/// </summary>

public Dictionary<Achievement, int> GetAchievementCounts(List<string> categories)

{

// Get the achievements of the specified categories

List<Achievement>achievements = GetAchievements(categories);

// Plonk these achievements into a dictionary used for the counts of these achievements

Dictionary<Achievement, int> counts = achievements.ToDictionary(x => x, x => 0);

string query;

foreach (Achievement x in counts.Keys)

{

// For each achievement, find the number of times it occurs in the achievementData

// table where it has been achieved and not just requested

query = "SELECT COUNT(`ID`) FROM `" + achievementDataTable + "` WHERE `Requested`='0'";

if (connectionOpen())

{

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

try

{

int count = 0;

// Execute the command and open the reader

OdbcDataReader dataReader = cmd.ExecuteReader();

dataReader.Read(); // Get the first record.

count = dataReader.GetInt16(0); // Read the first value in the record

dataReader.Close();

counts[x] = count;

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

}

return counts;

}

Web.config file addition

Finally, this line was added to the Web.config file. This defines the connection string used to define the OdbcConnection in the database class.

<add name="MySQLConnStr" connectionString="DRIVER={MySQL ODBC 3.51 Driver};Database=CAT;Server=localhost;UID=catAdmin;PWD=4XzPuba1aQPft6bV;"/>

### 1st December 2015

#### Further database work (App\_Code\Database.cs)

Work on the database class continued, with additional queries created. The first of which is GetUserAchievements, a query that returns a list of achievements that a user has obtained. After realising how long this function is, and how much some of its functionality would be re-used, I broke it down into three separate functions. GetUserAchievements, GetUserAchievementIDs and GetAchievementsFromIDs. The code for all three of these is below:

/// <summary>

/// Returns a list of ids of the achievements that the user has obtained

/// </summary>

public List<int> GetUserAchievementIDs(User user, bool requested, bool awarded)

{

// List of achievement ID's from the achievementData table

List<int> achievementIDs = new List<int>();

string query;

/\* Get all achievement ID's from the achievementData table for this user

\* that have pertain to the requested or awarded parameters. \*/

query = "SELECT `AchievementID` FROM '" + achievementDataTable + "' WHERE `UserID`='"

+ user.ID + "' AND `Requested`='" + Convert.ToInt16(requested) + "' AND `Awarded`='"

+ Convert.ToInt16(awarded) + "'";

if (connectionOpen())

{

// Creates a database command from the query and existing connection.

OdbcCommand cmd = new OdbcCommand(query, connection);

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// While there are more records to read...

while (dataReader.Read())

{

// Add the returned int to the achievementID list

achievementIDs.Add(dataReader.GetInt16(0));

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

return achievementIDs;

}

GetUserAchievementIDs, as seen above, takes the user, requested and awarded parameters, and pulls the ID’s of the achievements that the user has on the achievementData table. The requested and award parameters are used for further filtering of the results, and correlates directly to the columns in the table.

/// <summary>

/// Returns a list of achievements from a list of achievement id's

/// </summary>

public List<Achievement> GetAchievementsFromIDs(List<int> ids)

{

// List of achievements to store the achievements to return

List<Achievement> achievements = new List<Achievement>();

string query;

OdbcCommand cmd;

// For each of the returned IDs

foreach (int x in ids)

{

// Create a query to get that achievement's data from the achievement table

query = "SELECT \* FROM `" + achievementTable + "` WHERE `ID`='" + x + "'";

// Create a database command from the query and existing connection

cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// While there are more records to process...

while (dataReader.Read())

{

// Get the ID and make sure it is an actual achievement

int id = dataReader.GetInt16(0);

if (id >= 1)

{

// Then add the returned achievement to the achievement list

achievements.Add(new Achievement(id, dataReader.GetString(1),

dataReader.GetString(2), dataReader.GetString(3),

dataReader.GetString(4), dataReader.GetInt16(5),

dataReader.GetDateTime(6), dataReader.GetBoolean(7)));

}

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

}

return achievements; // Return the final list of acievments. May be empty if the

// query failed

}

/// <summary>

/// Returns a list of achievements that the user has obtained based on the

/// requested and awarded parameters.

/// </summary>

public List<Achievement> GetUserAchievements(User user, bool requested, bool awarded)

{

return GetAchievementsFromIDs(GetUserAchievementIDs(user, requested, awarded));

}

### 7th December 2015

#### Further database class queries (App\_Code\Database.cs)

Further work was completed on implementing the queries from the algorithms stage into the database class. Here is the first, GetUsers:

/// <summary>

/// Returns a list of all users in the system

/// </summary>

public List<User> GetUsers()

{

// Create a list to store the return values in

List<User> users = new List<User>();

// Create a query to return all users

string query = "SELECT \* FROM `" + userTable + "`";

// Create a command from the query and the existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// While there are more records to process...

while (dataReader.Read())

{

// Grab the ID and make sure it is an actual user...

int id = dataReader.GetInt16(0);

if (id >= 1)

{

// Then add the returned user to the user list

users.Add(new User(id, dataReader.GetString(1),

dataReader.GetString(3), dataReader.GetString(4),

dataReader.GetString(5), dataReader.GetBoolean(6),

dataReader.GetDateTime(7), dataReader.GetInt16(7),

dataReader.GetBoolean(8), dataReader.GetBoolean(9),

dataReader.GetInt16(10)));

}

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

return users; // Return the final list of users, may be empty if the query failed.

}

After looking further into the function GetUserScoresForCategories, I realised that there is a lot going on in this function that could be split into further functions, if for no other reason than readability. This meant that this part of the algorithm;

* 1. For each user
     1. Set the query to get the achievements ID’s that the user has obtained.
     2. Open the connection if it isn’t already
     3. Execute the query
     4. Open the data reader and get the return.
     5. Create a list of ID’s from the return.
     6. Close the data reader
     7. Get the total score that the achievements are worth using GetAchievementsScore
     8. Set the Integer in the original double dictionary for this user to that value.

Now becomes GetUserScoreForCategory, allowing other functions to also use this functionality. The code is below:

/// <summary>

/// Returns a KeyValuePair where the key is the user, and the value is the score for

/// the specified user and for the specified category, returned from the database

/// </summary>

public static int GetUserScoreForCategory(string category, User user)

{

// Get all achievments that the specified user has achieved and not just requested

string query = "SELECT `AchievementID` FROM `" + achievementDataTable +

"` WHERE `Requested`='0' AND `UserID`='" + user.ID + "'";

// Temporary list of achievement IDs used to store the data from the achievementData table

List<int> achievementIDs = new List<int>();

// Create a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open the reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// While there are more records to be read...

while (dataReader.Read())

{

// Add to the achievementID list the ID of the achievement just read

achievementIDs.Add(dataReader.GetInt16(0));

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

// Return the total worth of all of the specified achievements

return GetAchievementsScore(achievementIDs, new List<string> { category });

}

GetAchievementsScore was also coded, I did however, find a way to optimise the function. If you look at the algorithms stage, the function goes like this:

1. For each of the integers in the IDs list
   1. For each of the achievements in the achievement list
      1. If the integer and the ID of the achievement match,
         1. Add the value of the achievement to the score total.

And while this works, it is less efficient than the new solution which is:

1. For each achievement
   1. Check to see if the IDs list has the achievement’s ID
      1. If yes, add the value of the achievement to the score total

This results in less enumerating through the likely longer list (The achievement list), and means that the ID list is only looped through once. The code for this is as follows:

/// <summary>

/// Returns the total score from for the category from the specified achievment ID list

/// </summary>

public int GetAchievementsScore(List<int> achievementIDs, List<string> categories)

{

// Get all of the achievements from the database that have the specified category

List<Achievement> achievements = GetAchievements(categories);

int score = 0; // Variable that the sum of the achievements is to be stored in

// For each achievement..

foreach (Achievement achievement in achievements)

{

// See if the user has it.. and if it does..

if (achievementIDs.Contains(achievement.ID))

{

// Add the value of the achievement onto the total score

score += achievement.Value;

}

}

return score; // Return the total score

}

This all leaves the original function, GetUserScoresForCategories as follows:

/// <summary>

/// Returns a dictionary of categories, which contains a dictionary of users and their

/// scores in those categories. 07/12/15

/// </summary>

public Dictionary<string, Dictionary<User, int>> GetUserScoresForCategories(List<string> categories)

{

// Double dictionary to store values to be returned in organised manner

Dictionary<string, Dictionary<User, int>> scores = new Dictionary<string,

Dictionary<User, int>>();

// Get all of the users

List<User> users = GetUsers();

// For each category...

foreach (string cat in categories)

{

// Create a dictionary and..

scores.Add(cat, new Dictionary<User, int>());

// For each user...

foreach (User user in users)

{

// Add that user to the category along with their score for that category

scores[cat].Add(user, GetUserScoreForCategory(cat, user));

}

}

return scores; // Return the monolithic double dictionary of doom

}

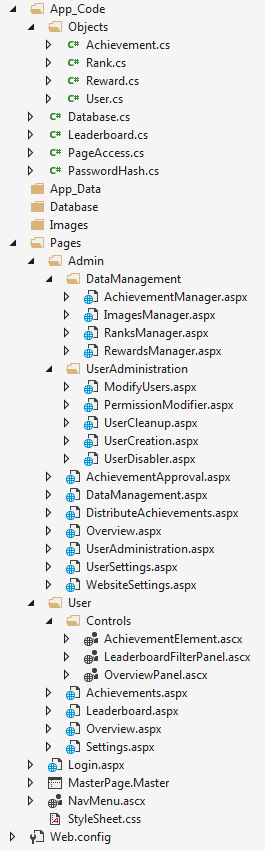
And with that, all of the database queries that were planned in the algorithms stage are complete. This by no means that there will be no further entries into the database class, but for now, it is done.

## Website work

### 10th – 15th December 2015

#### Creation of basic website items

The basic directory of the website was created as per the diagram on page 31 with a few additions.



In addition to this, the css stylesheet as defined on page 35 was copied over.

#### Creation of the Master Page (Pages/MasterPage.Master)

Before much more work can continue on the website, a consistent style must be defined. As mentioned just prior, the css stylesheet was copied over to the project, and that will help maintain a set style. However, another thing that needs to be created is the Master Page. The Master Page is a default style page for elements that will be consistent across multiple pages (E.g. Header, Navigation menu, license, etc.) I have already created my Master Page layout in the design stage, so replicating it wasn’t too difficult. In addition, from this point forward, html is going to start becoming as common as C#. Thought I’d mention that before continuing. The Master Page was defined as follows:

<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="MasterPage.Master.cs" Inherits="CAT\_2015.Pages.MasterPage" %>

<%@ Register TagPrefix="Controls" TagName="NavMenu" Src="~/Pages/NavMenu.ascx" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head id="Head1" runat="server">

<title></title>

<asp:ContentPlaceHolder ID="head" runat="server">

</asp:ContentPlaceHolder>

<link href="StyleSheet.css" rel="stylesheet" type="text/css" />

</head>

<body class="base">

<form id="form1" runat="server">

<div>

<asp:Table ID="table1" runat="server" Width="100%" CellPadding="0" CellSpacing="10">

<asp:TableRow>

<asp:TableCell Width="100%" Height="60px" CssClass="header"> <!-- Header -->

Computing Achievement Tracker

</asp:TableCell>

</asp:TableRow>

<asp:TableRow>

<asp:TableCell Width="85%" Height="100%" VerticalAlign="Top" CssClass="base"> <!-- Content -->

<asp:ContentPlaceHolder ID="ContentPlaceHolderContent" runat="server">

</asp:ContentPlaceHolder>

</asp:TableCell>

<asp:TableCell Width="15%" VerticalAlign="Top" Height="100%"> <!-- Navigation Menu -->

<Controls:NavMenu ID="NavMenu" runat="server" />

</asp:TableCell>

</asp:TableRow>

</asp:Table>

</div>

</form>

</body>

</html>

Which leads to a webpage looking like this (Once the navigation menu is created (That’s next))

#### Creation of the Navigation Menu (Pages/NavMenu.ascx)

Now it is worth mentioning that each element inside an ASP.NET website has what’s called a “codebehind” element. What this essentially means is there are two bits of code that make up each element. The html that defines simple things such as style and placement, and the codebehind which contains C# to be executed upon events occurring on the server. With this said, here is the html of the navigation menu:

<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="NavMenu.ascx.cs" Inherits="CAT\_2015.Pages.NavMenu" %>

<asp:Table ID="NavTable" runat="server">

</asp:Table>

As you can see, there really isn’t a lot. That’s because this literally just consists of a table that can be edited with codebehind. Here is the first codebehind function, for when the page loads:

CAT\_2015.User currentUser = null;

protected void Page\_Load(object sender, EventArgs e)

{

// Pull the current user via the client's sessionID

currentUser = AppCode.Database.GetLoggedInUser(Session.SessionID);

// If a user was found...

if (currentUser != null)

{

// Check if the user is an admin. If yes..

if (currentUser.PermissionLevel > 0)

{

// Add the buttons to the nav menu for admins

addButtons(AppCode.PageAccess.AdminPages);

}

else // If the user is just a normal user...

{

// Add the buttons to the nav menu for users

addButtons(AppCode.PageAccess.UserPages);

}

}

else

{

// If no user was found, add the default non-user pages

addButtons(AppCode.PageAccess.DefaultPages);

}

}

This function makes reference to a magical function called addButtons which I will now follow up with on the next page. addButtons uses the provided page dictionary, and adds buttons relating pages to page names to the navigation menu.

/// <summary>

/// Uses the specified links dictionary to add buttons to the naviation menu

/// </summary>

void addButtons(Dictionary<string, string> links)

{

// For each item in the dictionary..

foreach (string name in links.Keys)

{

// Create a row, cell and button

TableRow row = new TableRow();

TableCell cell = new TableCell();

Button button = new Button();

// Set the button's properties accordingly

button.ID = "button\_" + name;

button.CssClass = "button";

button.Text = name;

button.Click += buttonUser\_Click;

// And add the button, cell and row to the table

cell.Controls.Add(button);

row.Cells.Add(cell);

NavTable.Rows.Add(row);

}

}

Each button is being assigned the procedure buttonUser\_Click upon the Click event being raised. Let’s define that:

/// <summary>

/// Send the client to the correct page when the button is pressed

/// </summary>

void buttonUser\_Click(object sender, EventArgs e)

{

// Cast the sender as a button

Button button = (Button)sender;

// Log the user out if the button says logout

if (button.Text == "Logout")

{

AppCode.Database.RemoveSession(Session.SessionID);

Response.Redirect(CAT\_2015.AppCode.PageAccess.DefaultPages["Login"]);

}

else // Otherwise, redirect to the relavent page

{

if (currentUser != null) // If the user exists..

{

if (currentUser.PermissionLevel > 0) // If the user is an admin..

{

Response.Redirect(CAT\_2015.AppCode.PageAccess.AdminPages[button.Text]);

}

else // If the user is not an admin..

{

Response.Redirect(CAT\_2015.AppCode.PageAccess.UserPages[button.Text]);

}

}

else // If there is no user..

{

Response.Redirect(CAT\_2015.AppCode.PageAccess.DefaultPages[button.Text]);

}

}

}

With this procedure created, I make reference to two dictionaries in AppCode/PageAccess. I will now just quickly show how I defined those.

Three dictionaries were created. UserPages, AdminPages and DefaultPages. Each one contains the pages that will be displayed in the navigation menu for their privilege level. DefaultPages is for users not yet logged in. I will now show how these dictionaries were defined:

// Pages that are displayed on the navigation menu when a user is logged in

public static Dictionary<string, string> UserPages = new Dictionary<string, string>()

{

{"Home", "~/Pages/User/Overview.aspx"},

{"Overview","~/Pages/User/Overview.aspx"},

{"Leaderboard","~/Pages/User/Leaderboard.aspx"},

{"Achievements","~/Pages/User/Achievements.aspx"},

{"Settings","~/Pages/User/Settings.aspx"},

{"Logout",""}

};

// Pages that are displayed on the navigation menu when a admin is logged in

public static Dictionary<string, string> AdminPages = new Dictionary<string, string>()

{

{"Achievement Approval","~/Pages/Admin/AchievementApproval.aspx"},

{"Data Management","~/Pages/Admin/DataManagement.aspx"},

{"Distribute Achievements","~/Pages/Admin/DistributeAchievements.aspx"},

{"Overview","~/Pages/Admin/Overview.aspx"},

{"User Administration","~/Pages/Admin/UserAdministration.aspx"},

{"User Settings","~/Pages/Admin/UserSettings.aspx"},

{"Website Settings","~/Pages/Admin/WebsiteSettings.aspx"},

{"Logout",""}

};

// Pages that are available without the need to log in

public static Dictionary<string, string> DefaultPages = new Dictionary<string, string>()

{

{"Home", "~/Pages/Home.aspx"},

{"Login", "~/Pages/Login.aspx"}

};

Note that these are not final and will be modified in the future.

Additionally, right on the first function for the navigation menu, Page\_Load, I made a reference to the database class and called a function called GetLoggedInUser. This is part of the authentication system, which I will now go into depth about.

#### First attempt at creation of the Authentication system

There is one elephant in the room at this point: How is the system supposed to know which user is logged in where when each page is counted as a separate request? At first I thought the answer was cookies, and I’d like to delve into my short-lived solution for a second.

As a sort of ‘remember me’ system, I was going to give each user a unique identifier upon logging in, which would be stored in the database alongside their user. This unique identifier would be their ‘SessionID’ which would also be stored in their local cookies to store the current session. To increase security, I was going to make this cookie expire after a day, and this is the code that I produced for this solution:

const int sessionIDLength = 16;

/// <summary>

/// Generates a sessionID using a random string generator, with the username

/// appended to ensure it is unique.

/// </summary>

public static byte[] generateSessionID(string username)

{

// Random byte provider

RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

// Create a sessionID of length decided by the above constant.

byte[] sessionID = new byte[sessionIDLength];

// Create a byte array from the username

byte[] unBytes = Encoding.ASCII.GetBytes(username);

// Get random bytes using the random byte provider, and add these to the sessionID.

rng.GetBytes(sessionID);

// Create a final array that is as long as both arrays put together

byte[] finalArray = new byte[sessionID.Length + unBytes.Length];

// Put the arrays together and return the result

sessionID.CopyTo(finalArray, 0);

unBytes.CopyTo(finalArray, sessionID.Length);

return finalArray;

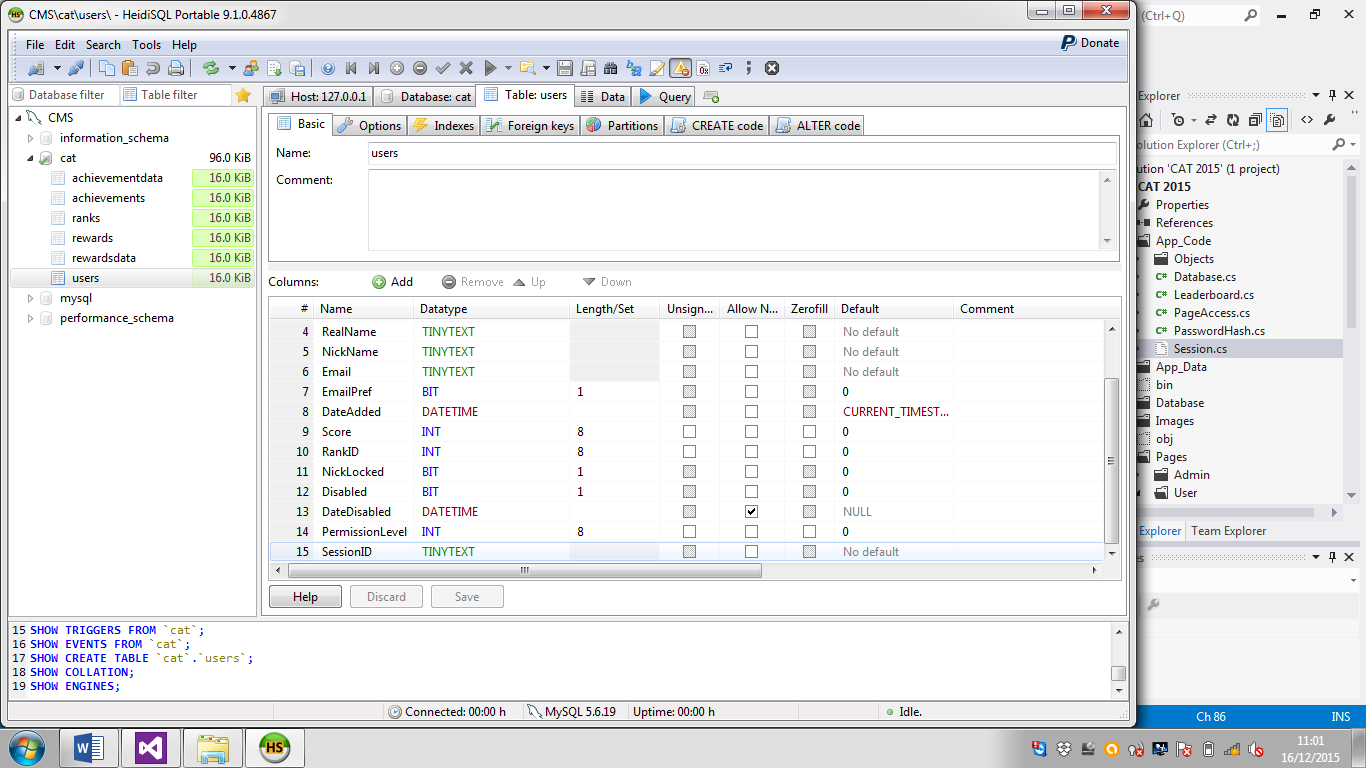
}

The exact function of this code was to produce a unique identifier using a cryptographically secure random generator, and to ensure that each one was unique, append the username onto it. This had to be generated using a cryptographically secure generator in order to stop sessions from being able to be generator by users with mal intentions. This code was later ditched however, due to me discovering that ASP.NET gives each user a unique SessionID by default anyway, but I am proud of this solution nonetheless.

#### Actual Authentication System implementation

##### Database Edits

So the actual implementation of the authentication system ended up being less complicated. It still used the same underlying principles, but without the complicated SessionID generation. With that being said, the first edit required an edit to the database, and more specifically, the user table:



The SessionID column was created, in order for the SessionID’s to be stored beyond a single page’s codebehind file. With this being created, the existing database functions defined earlier had to be modified, which is where I made a slight maintenance tweak to the database class (App\_Code/Database.cs). Instead of writing how a user should be read on every single query, I made a new function readUser which took in the datareader as a parameter, read a user and returned the user. This makes further modifications to this table easier to implement. The code for this function is as follows:

/// <summary>

/// Constructs a user from the output of a datareader. Assumes that there is data

/// ready to be read from the current record 14/12/15

/// </summary>

private static User readUser(OdbcDataReader dataReader)

{

return new User(dataReader.GetInt16(0), dataReader.GetString(1),

dataReader.GetString(3), dataReader.GetString(4), dataReader.GetString(5),

dataReader.GetBoolean(6), dataReader.GetDateTime(7), dataReader.GetInt16(8),

dataReader.GetInt16(9), dataReader.GetBoolean(10), dataReader.GetBoolean(11),

dataReader.GetInt16(13));

}

Which leads to previous functions, such as get users having this instead:

if (id >= 1)

{

// Then add the returned user to the user list

users.Add(readUser(dataReader));

}

The same was done for the other data types that could be returned (Rank, Reward, Achievement). With these database edits said and done, the next thing was to create a login page and the associated procedures for logging in.

##### Logging in (Pages/Login.aspx)

The first thing that needs completing is the login page, which will use a function like this, to achieve it’s logging in procedure.

1. Get the current user
2. If the current user is not logged in…
   1. Perform a validation check on the textboxes
   2. If the input is valid…
      1. Verify the user and password with the database
      2. If the username and password are correct…
         1. Set the logged in user to this user, and update this user’s SessionID to the current one
         2. And redirect the user to their homepage

Which ended up leading to the login page’s codebehind file having three functions. Page\_Load, buttonLogin\_Click and redirectHome. Page\_Load dictates what happens when the page loads, buttonLogin\_Click performs the above algorithm and redirectHome sends the user to their appropriate home page. Here is the variable definitions and Page\_Load:

// The current session's associated user

CAT\_2015.User currentUser = null;

protected void Page\_Load(object sender, EventArgs e)

{

// Get the current session's user and set currentUser to it

currentUser = AppCode.Database.GetLoggedInUser(Session.SessionID);

buttonLogin.Click += buttonLogin\_Click;

}

Here is the redirectHome procedure:

/// <summary>

/// Take the user to their appropriate home page

/// </summary>

private void redirectHome()

{

// If there is a current user logged in...

if (currentUser != null)

{

// If the user is an admin...

if (currentUser.PermissionLevel > 0)

{

Response.Redirect(System.Configuration.ConfigurationManager.AppSettings["AdminHome"]);

}

else

{

Response.Redirect(System.Configuration.ConfigurationManager.AppSettings["UserHome"]);

}

}

}

And finally, on the next page is buttonLogin\_Click.

void buttonLogin\_Click(object sender, EventArgs e)

{

// If there is no user logged in...

if (currentUser == null)

{

// Make sure there is something in the textboxes..

if (textBoxUsername.Text.Length >= 1 && textBoxPassword.Text.Length >= 1)

{

// If the password is correct for the user (if the user exists)..

if (CAT\_2015.AppCode.Database.AuthenticateUser(

textBoxUsername.Text.ToUpper(),

textBoxPassword.Text, Session.SessionID))

{

// Set the current user to that of the user that was logged in,

// and set that user's sessionID to this client's sessionID

currentUser = AppCode.Database.GetLoggedInUser(Session.SessionID);

// Then take the user to their appropriate home page

redirectHome();

}

else

{

labelError.Text = "Could not authenticate user. Please check your "

+ "username and password and try again. The user could also no "

+ "longer be avaialable.";

}

}

else

{

labelError.Text = "Please enter in a valid username and password.";

}

}

else

{

// Take the user to their appropriate home page

redirectHome();

}

}

In order for redirectHome to work and make it easily maintainable, I added two extra entries to the web.config file, AdminHome and UserHome. These variables store the webpages that have been designated as the user and admin’s home pages. The code for this is below:

<appSettings>

<add key="AdminHome" value="~/Pages/Admin/Overview.aspx"/>

<add key="UserHome" value="~/Pages/User/Overview.aspx"/>

</appSettings>

And while we’re here, here’s the HTML that makes up the Login page:

<%@ Page Title="Computing Achievement Tracker - Login" Language="C#" MasterPageFile="~/Pages/MasterPage.Master" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="CAT\_2015.Pages.Login" %>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolderContent" runat="Server">

<asp:Table Width="100%" HorizontalAlign="Center" runat="server" CssClass="base" CellSpacing="6">

<asp:TableRow CssClass="base">

<asp:TableCell CssClass="base">

Username:

</asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="base">

<asp:TableCell>

<asp:TextBox ID="textBoxUsername" runat="server" CssClass="textbox"></asp:TextBox>

</asp:TableCell>

</asp:TableRow>

<asp:TableRow cssClass="base">

<asp:TableCell CssClass="base">

Password:

</asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="base">

<asp:TableCell>

<asp:TextBox ID="textBoxPassword" runat="server" CssClass="textbox" TextMode="Password"></asp:TextBox>

</asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="base">

<asp:TableCell>

<asp:Button ID="buttonLogin" runat="server" CssClass="button" Text="Login"></asp:Button>

</asp:TableCell>

</asp:TableRow>

<asp:TableRow>

<asp:TableCell>

<asp:Label ID="labelError" runat="server" CssClass="base"></asp:Label>

</asp:TableCell>

</asp:TableRow>

</asp:Table>

</asp:Content>

##### Storing Passwords Securely (App\_Code/PasswordHash.cs)

Since I will be storing usernames, email addresses and passwords, precautionary steps should be taken to protect the user’s data. The MySQL database that will be used when the system is deployed will only have local access, but just in case a user with malicious intentions does gain access to the database, passwords must not be stored in plain text, and I will therefore be using an industry standard: PBKDF2.

The website crackstation.net has an excellent article on salting and hashing passwords and why it is required, but also class files for .NET that can be used to store passwords securely, and it is this that I shall be using. I won’t bother documenting this code, as it is not my own, but the class shall be added as App\_Code/PasswordHash.cs. The primary two functions I shall be using from this class will be CreateHash and ValidatePassword. CreateHash creates a salt and hash from the password and returns it as a single string. ValidatePassword takes a password and the correct hash and returns whether the two match. It does not however, give the correct password unless it is already known.

With all this said and done, it is time to implement these functions as part of the authentication system as database queries.

##### Authentication database queries (App\_Code/Database.cs)

The first database query that needs to be created is AuthenticateUser. The pseudo-code for this function is below:

1. Create a query to get the salted and hashed password as well as the disabled state of the user for the current user from the users table
2. Open the connection is it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Close the data reader
6. If the user is not disabled…
   1. If the password successfully validates against the salted and hashed version…
      1. Update this user’s SessionID with the current SessionID.
      2. Return true
7. If something failed, return false

And the code for this is as follows:

/// <summary>

/// Used to login the user. Uses the username and password provided and hashes the

/// password, checks it against the existing password and authenticates, if

/// successful, returns true with UserID out for cookie storage of user session.

/// </summary>

public static bool AuthenticateUser(string username, string password, string sessionID)

{

// Pull the password from the database for the specified user

string query = "SELECT `sahp`, `disabled` FROM `users` WHERE `Username`='" + username + "'";

string sahp = "";

bool disabled = true;

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// If the query has returned anything...

if (dataReader.HasRows)

{

// Advance to first row

dataReader.Read();

// Get the salted and hashed password

sahp = dataReader.GetString(0);

disabled = dataReader.GetBoolean(1);

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

// Checks that the user has not been disabled

if (!disabled)

{

// Takes the salted and hashed password and uses it to validate the provided password

if (PasswordHash.ValidatePassword(password, sahp))

{

// If the password is correct, sets the SessionID in the database to

// the current client's SessionID and returns that authentication was sucessful

ExecuteNonQuery("UPDATE `Users` SET `SessionID`='" + sessionID +

"' WHERE `Username`='" + username + "';");

return true;

}

}

}

// If something didn't pass, authentication was unsucessful

return false;

}

The second function that needs to be created is GetLoggedInUser. This takes the current SessionID and figures out which user it belongs to, then returns that user so webpages can display data relevant to that user. The layout for this function is as follows:

1. Create a query to find any users with the SessionID specified and return all of that user’s data (Except password).
2. Open the connection if it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Close the data reader
6. Return the user, if any.

And here is the code:

/// <summary>

/// Returns the logged in user using the sessionID. 14/12/15

/// </summary>

public static User GetLoggedInUser(string sessionID)

{

// Find a user with the current client's sessionID

string query = "SELECT \* FROM `Users` WHERE `SessionID`='" + sessionID + "'";

User user = null;

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// Advance to the first row

dataReader.Read();

// If the query has returned a user...

if (dataReader.HasRows)

{

// Get the user

user = readUser(dataReader);

}

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

// Returns the constructed user, or null if something bad occurred

return user;

}

And that will do for the authentication system for now. (I’m aware that there is currently no way to create users but that will come later with the admin system).

#### User Overview Screen

We’re finally onto - arguably - the most important page of the entire system, the user overview page. This page will be a quick go-to page for statistics for the user. Going back to the original design, the first thing that needs to be created is the Overview Panel Control. I shall start this by designing the layout and working in the first basic bits of information that can be directly pulled from the user class. The html for the overview panel is below.

<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="OverviewPanel.ascx.cs" Inherits="CAT\_2015.Pages.User.Controls.OverviewPanel" %>

<asp:Table ID="overviewTable" runat="server" CssClass="base" Width="100%" Height="100%" CellSpacing="0">

<asp:TableRow CssClass="table">

<asp:TableCell CssClass="table" Width="200px"><asp:Label ID="labelUsername" runat="server" CssClass="overviewPanel"></asp:Label></asp:TableCell>

<asp:TableCell CssClass="table" Width="250px"><asp:Label ID="labelNickname" runat="server" CssClass="overviewPanel"></asp:Label></asp:TableCell>

<asp:TableCell CssClass="table" Width="200px"><asp:Label ID="labelActualname" runat="server" CssClass="overviewPanel"></asp:Label></asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="table">

<asp:TableCell RowSpan="2" CssClass="table" Width="200px"><asp:Image ID="imageRank" runat="server" /></asp:TableCell>

<asp:TableCell CssClass="table" Width="250px"><asp:Label ID="labelRankName" runat="server" CssClass="overviewPanelImportant"></asp:Label></asp:TableCell>

<asp:TableCell RowSpan="2" CssClass="table" Width="200px"><asp:Label ID="labelTotalPoints" runat="server" CssClass="overviewPanelImportant"></asp:Label></asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="table">

<asp:TableCell CssClass="table" Width="250px"><asp:Label ID="labelRankNumber" runat="server" CssClass="overviewPanelImportant"></asp:Label></asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="table" Width="100%">

<asp:TableCell CssClass="table" Width="100%" Height="30px" ColumnSpan="3"><!-- Progress bar goes here --></asp:TableCell>

</asp:TableRow>

<asp:TableRow CssClass="table">

<asp:TableCell CssClass="table" Width="100%" ColumnSpan="3"><asp:Label ID="labelProgressPercentage" runat="server" CssClass="overviewPanel"></asp:Label></asp:TableCell>

</asp:TableRow>

</asp:Table>

And this produces the exact layout that was designed previously for the overview panel. The only thing missing is the progress bar, as I have yet to create that as a control. So the first thing that needs doing is to pull the current user from the database, and then ensure that this user is a user and not just null. Once that is done, the labels should be filled with the relevant information. That being said, the code used to do these things is on the next page.

CAT\_2015.User currentUser = null; // Current client's associated user

protected void Page\_Load(object sender, EventArgs e)

{

// Get the logged in user

currentUser = AppCode.Database.GetLoggedInUser(Session.SessionID);

if (currentUser != null) // If a user was found..

{

// Then fill the labels with the relavent information..

labelUsername.Text = "Username: " + currentUser.Username;

labelNickname.Text = "Nickname: " + currentUser.NickName;

labelActualname.Text = "Real Name: " + currentUser.RealName;

labelTotalPoints.Text = currentUser.Score.ToString() + " Points";

// If the nickname is locked..

if (currentUser.NickLocked)

{

// Make it red and display (Locked)

labelNickname.CssClass = "nickLocked";

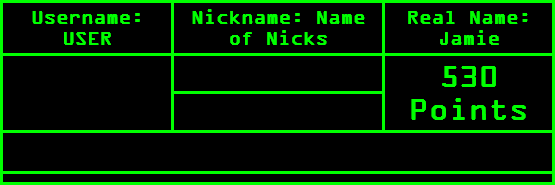
labelNickname.Text += " (Locked)";

}

}

}

This produces an overview panel that looks like this when running:



The next thing to consider is rank information, which makes up the rest of the overview panel. In order to achieve this, the current Rank and the next Rank for the User must be found. This will change the overall function for the overview panel to:

1. Get the current user, if a user is found…
   1. Fill out user-related information
   2. Get the current Rank, if rank is found…
      1. Fill out current Rank related information
      2. Get the next rank, if the next rank is found…
         1. Fill out next rank related information
      3. If no next rank is found, assume max rank and display this in the place of the progress percentile. (This will be changed)

The new code for the overview panel is now on the next page, however, the progress bar is still not included.

CAT\_2015.User currentUser = null; // Current client's associated user

CAT\_2015.Rank currentRank = null; // Current user's current rank

CAT\_2015.Rank nextRank = null; // Current user's next rank

protected void Page\_Load(object sender, EventArgs e)

{

// Get the logged in user

currentUser = AppCode.Database.GetLoggedInUser(Session.SessionID);

if (currentUser != null) // If a user was found..

{

// Get the user's current rank

currentRank = AppCode.Database.GetRankFromID(currentUser.RankID);

// And fill the labels with the relavent information..

labelUsername.Text = "Username: " + currentUser.Username;

labelNickname.Text = "Nickname: " + currentUser.NickName;

labelActualname.Text = "Real Name: " + currentUser.RealName;

labelTotalPoints.Text = currentUser.Score.ToString() + " Points";

// If the nickname is locked..

if (currentUser.NickLocked)

{

// Make it red and display (Locked)

labelNickname.CssClass = "nickLocked";

labelNickname.Text += " (Locked)";

}

if (currentRank != null) // If a current rank was found..

{

// Get the next rank

nextRank = AppCode.Database.GetNextRankUp(currentRank);

// And fill out information relating to the current rank

labelRankName.Text = currentRank.Name;

labelRankNumber.Text = "Level " +

AppCode.Database.GetRankRanking(currentRank).ToString();

imageRank.ImageUrl = currentRank.Image;

imageRank.ImageAlign = ImageAlign.Middle;

if (nextRank != null) // If the user's next rank was found..

{

int percentage =

Convert.ToInt16(((double)(currentUser.Score - currentRank.MinScore) /

(double)(nextRank.MinScore - currentRank.MinScore)) \* 100);

// Fill the labels with relavent information

labelProgressPercentage.Text = "Next Level: " +

(currentUser.Score - currentRank.MinScore).ToString() +

" / " +

(nextRank.MinScore - currentRank.MinScore).ToString() +

" ( " +

percentage.ToString() +

"% ) ";

}

else

{

labelProgressPercentage.Text = "Max Level Achieved!";

}

}

}

}

##### Database Edits

In the new overview panel, I am using several functions that were added to the database class (AppCode/Database.cs) that has not been documented as of yet (GetRankFromID, GetNextRankUp, GetRankRanking), so I shall go through each one now.

GetRankFromID Takes an ID, and finds the rank with that ID and returns it as a Rank object. The layout for GetRankFromID is as follows:

1. Create a query to find the rank with the user’s RankID.
2. Open the connection if it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Close the data reader
6. Return the rank, if any.

The actual code for this function is below:

/// <summary>

/// Returns the rank with the associated id

/// </summary>

public static Rank GetRankFromID(int id)

{

// Find the rank associated with the provided id

string query = "SELECT \* FROM `ranks` WHERE `ID`='" + id + "'";

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

Rank rank = null;

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// If the query has returned any results at all..

if (dataReader.HasRows)

{

// Get the next record

dataReader.Read();

// Read the record as a rank

rank = readRank(dataReader);

// Close the data reader

dataReader.Close();

}

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

// Returns the constructed rank, or returns null if something bad happened

return rank;

}

The next function is GetNextRankUp. This function takes the current rank, and using the MinScore property, obtains the next rank up from the current rank, and returns it as a rank object. The layout for this function is as follows:

1. Create a query to find the first rank with a higher MinScore than the current rank
2. Open the connection if it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Close the data reader
6. Return the rank, if any.

The code for this function is below:

/// <summary>

/// Returns the rank above the supplied rank

/// </summary>

public static Rank GetNextRankUp(Rank currentRank)

{

// Find the first rank with a higher MinScore than the current rank

string query = "SELECT \* FROM `ranks` WHERE `MinScore`>'" + currentRank.MinScore +

"' ORDER BY `MinScore` ASC LIMIT 1";

// Creates a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

Rank rank = null;

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// If the data reader returns anything

if (dataReader.HasRows)

{

// Read the first record

dataReader.Read();

// Read the record as a rank and add it to the rank list

rank = readRank(dataReader);

}

// Close the data reader

dataReader.Close();

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

}

return rank;

}

The final function for this section is GetRankRanking. This function takes the current rank, and using the MinScore property, obtains the number of ranks below it, and therefore the ranks ranking, and returns it as an int. The layout for this function is as follows:

1. Create a query to get the number of ranks where the MinScore is below the current rank's MinScore
2. Open the connection if it isn’t already
3. Execute the query
4. Open the data reader and get the return
5. Close the data reader
6. Return the ranking, will return 1 if no ranks found below the current one.

And here is the code for this function:

/// <summary>

/// Returns the ranking of the current rank. Example: Would return 1 for Inigma, assuming

/// default dataset.

/// </summary>

public static int GetRankRanking(Rank rank)

{

// Get the number of ranks where the MinScore is below the current rank's MinScore

string query = "SELECT COUNT(0) FROM `Ranks` WHERE `MinScore`<'" + rank.MinScore + "'";

int ranking = 0;

// Create a database command from the query and existing connection

OdbcCommand cmd = new OdbcCommand(query, connection);

if (connectionOpen())

{

try

{

// Execute the command and open a reader

OdbcDataReader dataReader = cmd.ExecuteReader();

// If any data was returned by the reader..

if (dataReader.HasRows)

{

// Advance to first record

dataReader.Read();

// Read the ranking

ranking = dataReader.GetInt16(0);

// Close the reader

dataReader.Close();

}

}

catch (OdbcException ex)

{

// Displays an error if something bad occurs while executing the command

error = ex.Message;

}

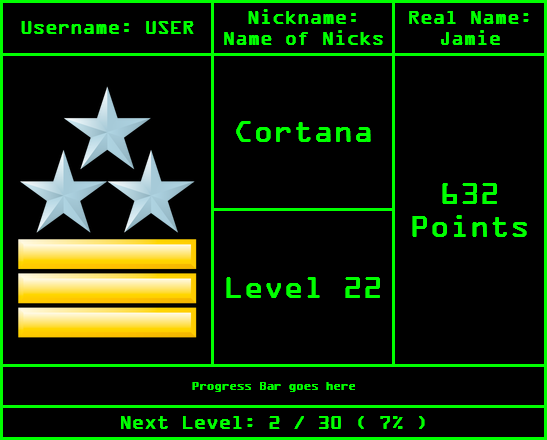
}

// Return the number of ranks with lower minScore + 1 for ranking position

return ranking + 1;

}

And all of this produces an overview panel that looks like this:



### 4th January

#### First presentation to end user

It was at this point that I decided to show the system to the end user.

# D. Documentation

# E. Evaluation

## E. i) Discussion of the degree of success in meeting the original objectives

## E. ii) Evaluate the user’s response to the system

## E. iii) Desirable Extensions

# Project Appendixes