MineService

CJ Miller

Collin Trowbridge

Brandon Hennessey

3rd February 2016

Team D

Milestone 4

Table of Contents

Introduction

MineService is a Client-Server application that would help monitor and run Minecraft servers across different systems. It can be split into two different feature sets. One for dealing with physical systems, such as resource allocation, while the other dealing with people, such as support tickets. Between these two sets there will be many sub features.

For physical system features the main ones will be on control. We'll have ways to start, stop, and restart the server and also automated versions of those. There will also be ways to upload, edit, and delete files on the server.

The people system will include both support and permissions systems. The support side will allow people to both submit and resolve issues in a centralized semi-public place. The permissions system will control who has access to what physical system features and what things they can see on the support side.

CSSE371 Change Log

Milestone 7:

Added business model canvas

Added value proposition canvas

Reported status of changes based on usability lab

Milestone 6:

Added pictures for new completed features.

Updated pictures for some completed pictures.

Milestone 5:

Updated the high-fidelity prototype images that changed.

Added "Feature Completed" section.

Milestone 4:

There has been no significant changes since Milestone three. We have implemented the changes our client suggested and still working on making some of the pages more to their liking.

User Stories

Epic

Performance

- As a developer I want to give typical users better performance when running Minecraft users so that they can have a more enjoyable Minecraft experience.

Time Saving

- As a developer I want to give typical users the ability to minimize their time spent on maintenance activities so that they can have their server up and running quicker.

Ease of Use

- As a developer I want typical users to be able to easily accomplish their everyday tasks so that they do not have to waste time looking up fixes or syntax.

Feature

Performance

- **VM Overhead**
  - As a Server Administrator I want to reduce CPU and Memory Usage so that I can dedicate more to my clients.
  - Condition of Satisfaction:
    - Drastically reduce the number of required VMs per client.
- **Memory Limit**
  - As a Server Administrator or Minecraft Admin I want to be notified when a Minecraft Server is approaching its memory limit so that I can see if it is an issue or just user spike
  - Condition of Satisfaction:
    - I am notified via text or email within a minute of a condition being met.
- **Frozen Servers**
  - As a Server Administrator or Minecraft Admin I want to be notified when a Minecraft Server is frozen so that I can fix it promptly
  - Condition of Satisfaction:
    - I am notified via text or email within a minute of a condition being met.

Ease of Use

- **Terminal**
  - I'm a person who is responsible for a couple of Minecraft servers but I don't like to program. I prefer not having to memorize code or commands and type it in the terminal because there are so many of

them. To make my life easier, I need to have a User Interface which would do all the stuff the terminal commands do with just a click of a button.

- Condition of Satisfaction:
  - Acknowledge that I don't have to use terminal to manage my servers

- **Multiple Applications Open**
  - I've realized that the less number of applications I have open, the less CPU usage and memory leaks. Having multiple applications open can potentially slow down the servers and is also really hard to manage since I need to switch between them at the right time for the right thing. It would be so much better if I just had to have one application open which would display all the information which I seek from multiple applications and perform an action based on that.
  - Condition of satisfaction:
    - Just one User Interface which would display all the information needed

Time Saving

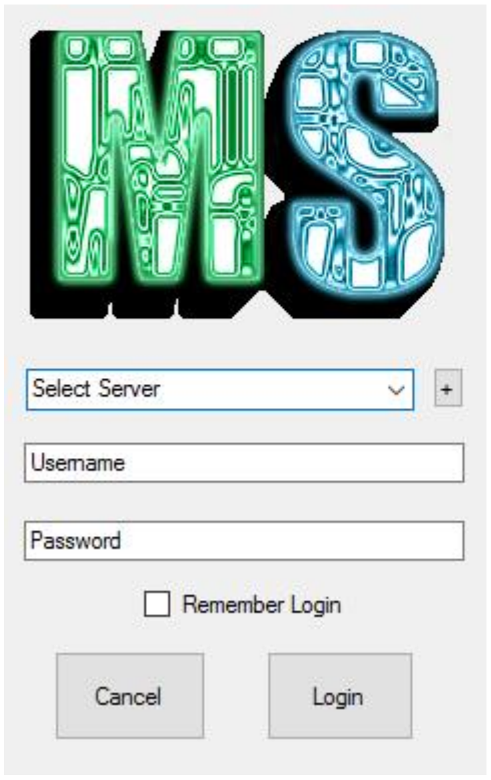
- **Terminal Commands have Multiple Steps**
  - I'm an individual who owns a few Minecraft servers, however I don't write programs as my profession. I would rather not have to enter commands into a terminal, because it takes time to memorize the commands, and if I don't memorize the procedure of how to navigate the terminal, I will have to take time to write a document with steps outlining the commands I have to execute. It would save me time if I only had to click a button on a graphical user interface, and all of the terminal commands would be done for me.
  - Conditions of satisfaction:
    - Make sure that I don't have to enter any commands into a terminal to manage my server.

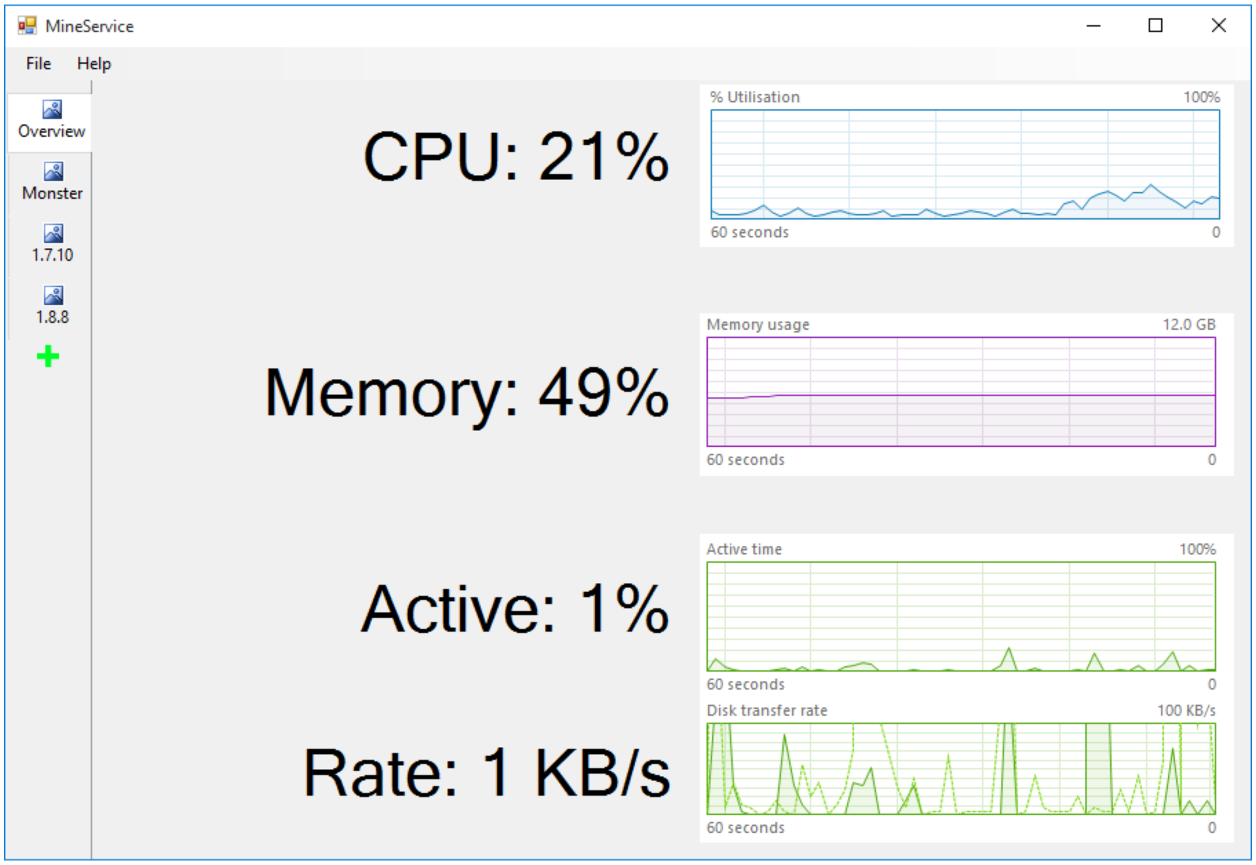
- **Regularly having to login to Check the Servers**
  - I've been told that Minecraft servers with a lot of mods on them can cause memory leaks, and memory leaks can cause a lot of memory, disk, and possibly CPU strain on my server. This can slow down all of my servers, and possibly cause them to crash. Therefore, I must log in to the windows application every three days or so to check the memory, disk, and CPU usage of each server. If a server is taking up too much hardware, I have to hope that no one is on the server, so that I can stop the server, then start it again. If there are people on the server, I either have to stop the server anyway, which will kick off all of the people who are on the server, or I have to wait until another time, log in again, and check and see if anyone is on the server. It would be much more

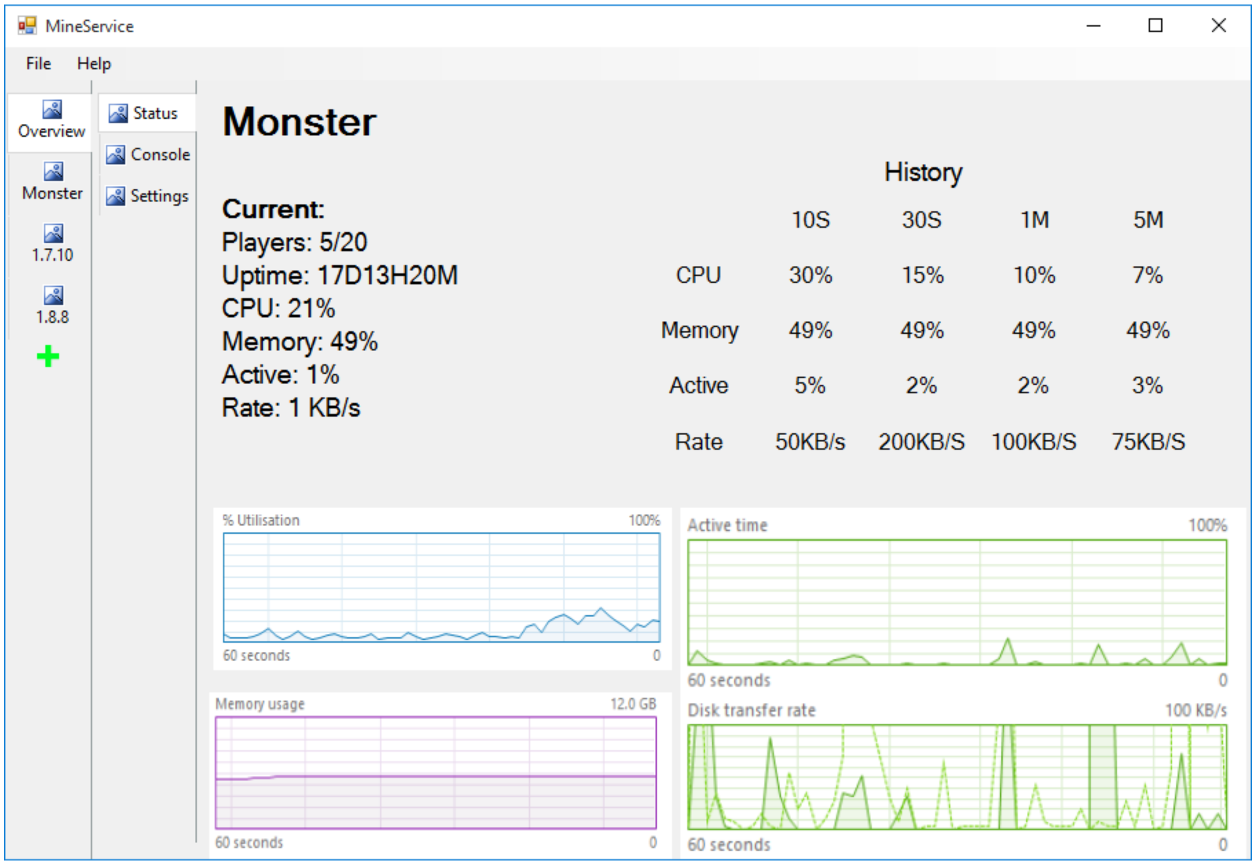
convenient if a server automatically restarted if no players were on and the combination of CPU, memory, and disk usage reached a certain level. Also, if the server was not able to be automatically restarted, due to players being on the server, and CPU, disk, or memory usage got too high, I would like to receive a text message, e-mail, or both, describing the issue to me, so that I can manually restart the server if I saw the need.

- Conditions of satisfaction:
    - If there are players on a server, and the combination of CPU, memory, and disk usage exceeds a certain level, notify the server owner via their specified means of contact, either text message, e-mail, or both.
    - Have the server automatically restart if the combination of CPU, memory, and disk usage exceds a certain level, and there are no players on the server.

Low-Fidelity Prototypes







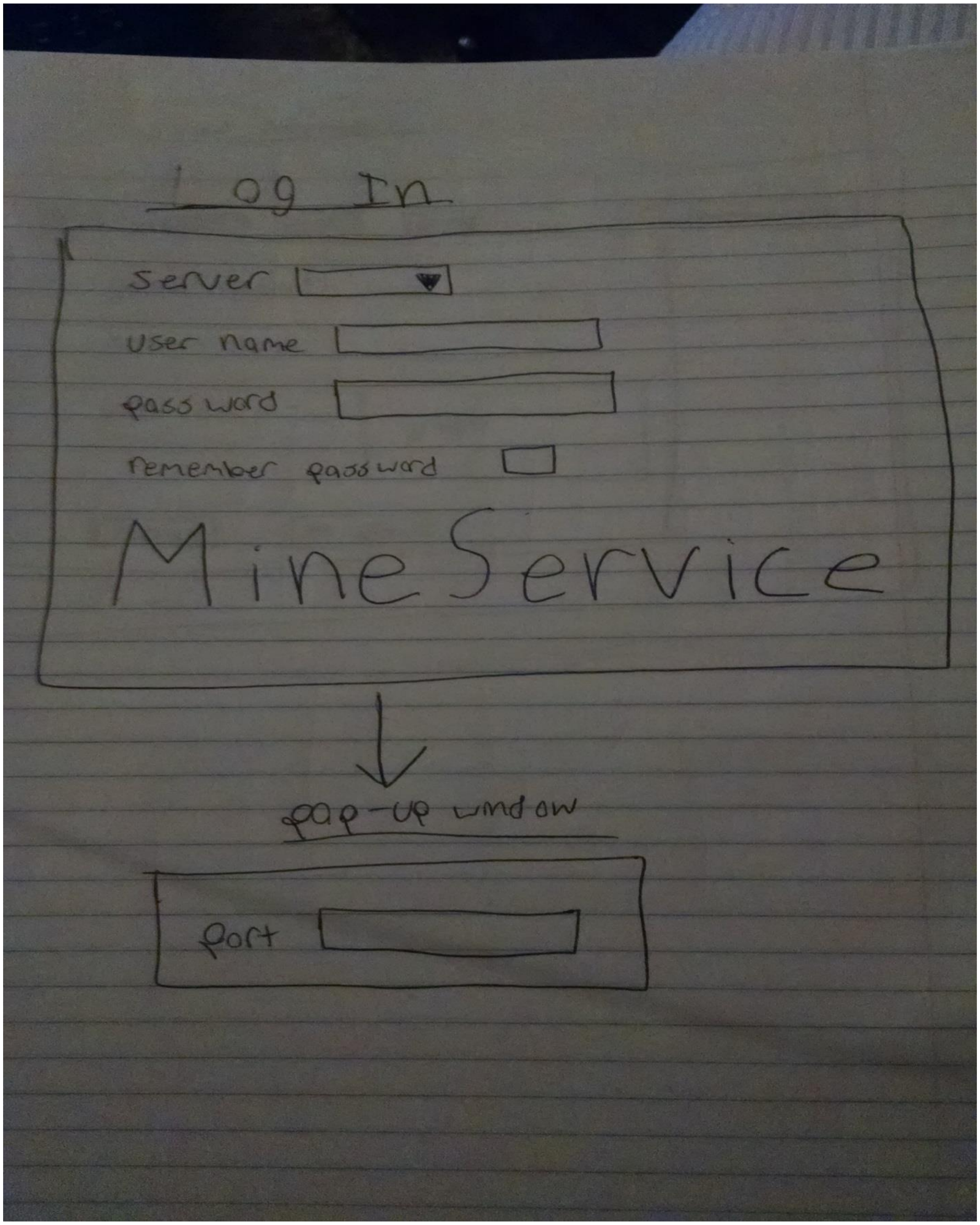
Monster

Current:
Players: 5/20
Uptime: 17D13H20M
CPU: 21%
Memory: 49%
Active: 1%
Rate: 1 KB/s

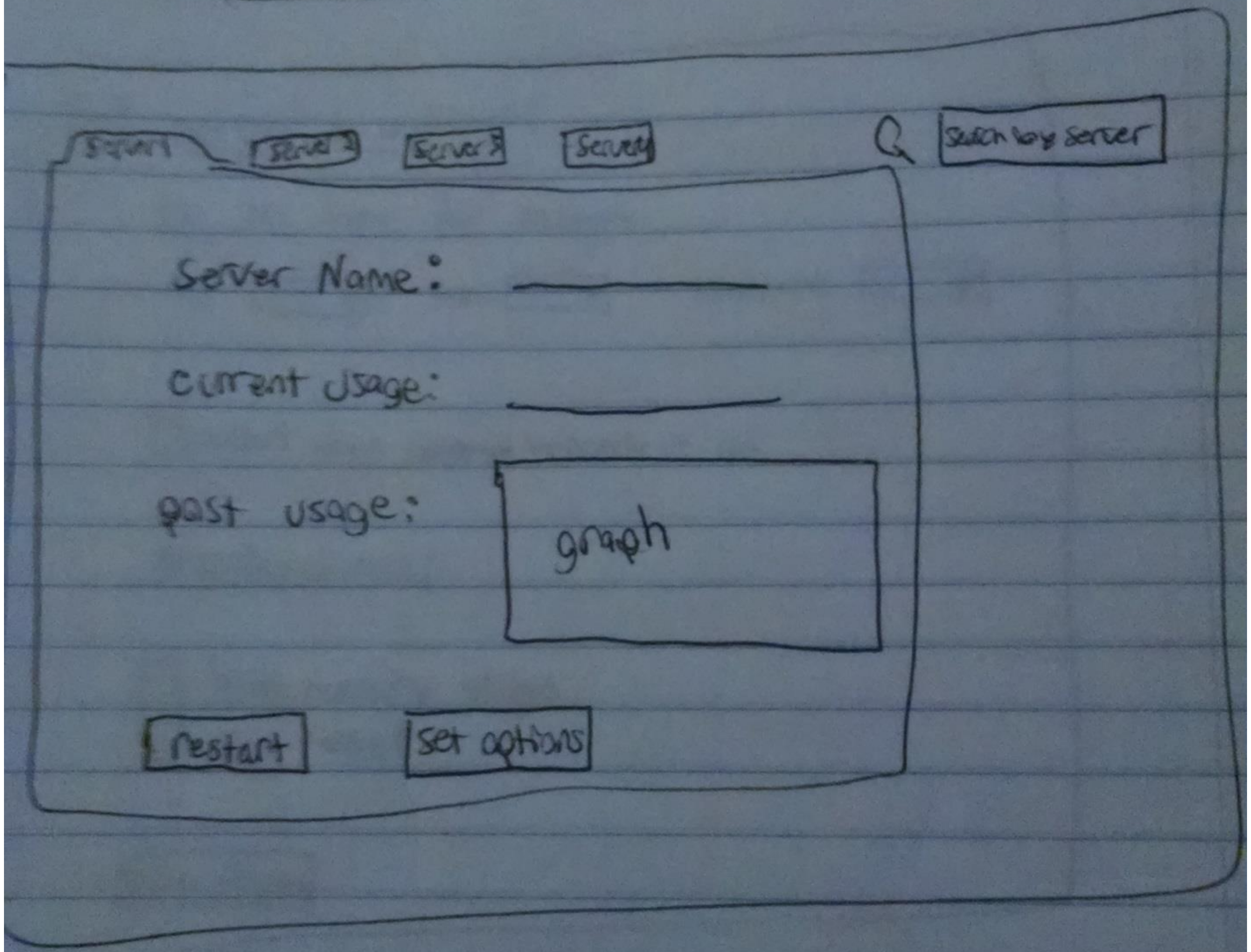
History

	10S	30S	1M	5M
CPU	30%	15%	10%	7%
Memory	49%	49%	49%	49%
Active	5%	2%	2%	3%
Rate	50KB/s	200KB/S	100KB/S	75KB/S

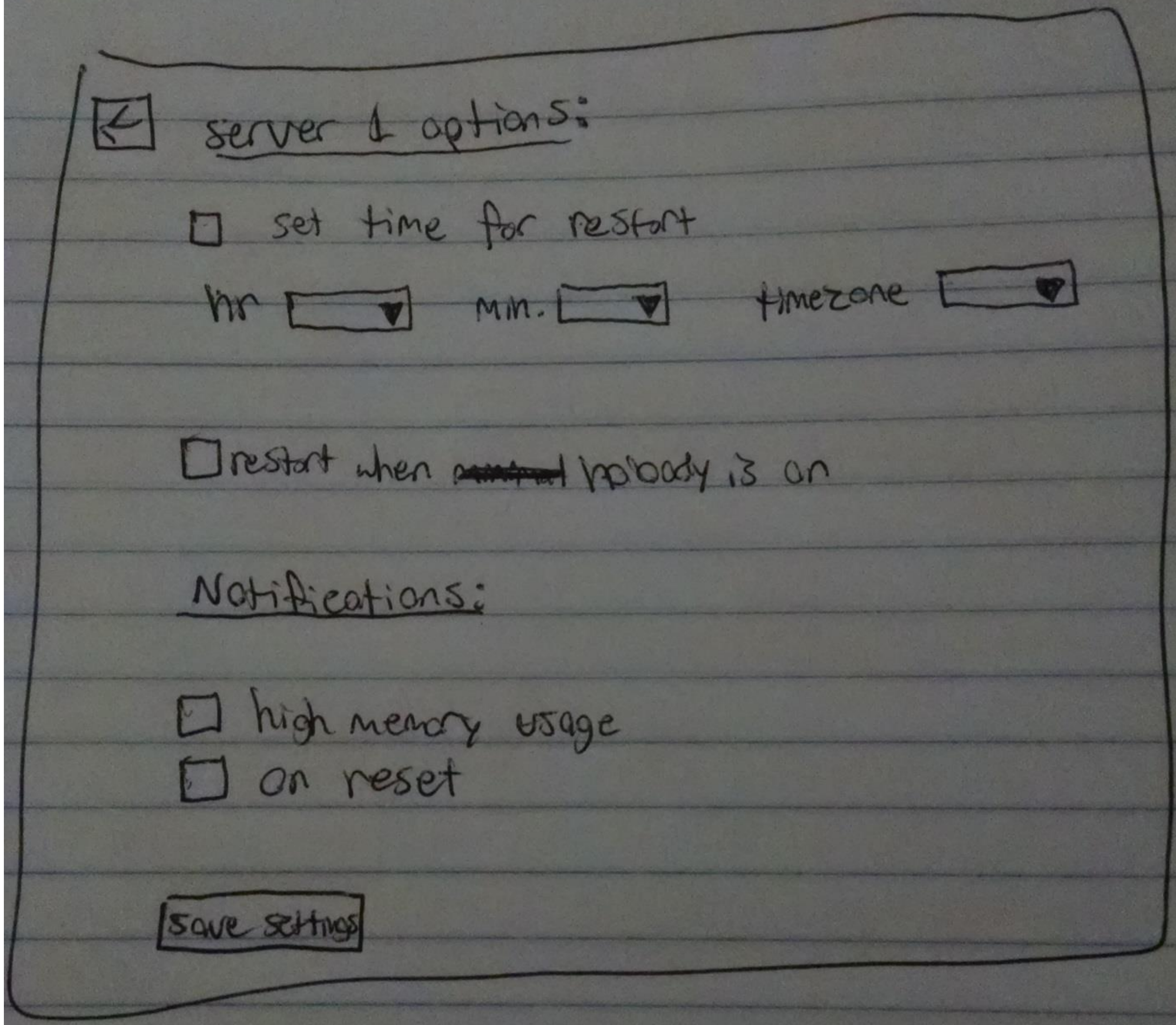
Alternative Design



Server information



Options Screen



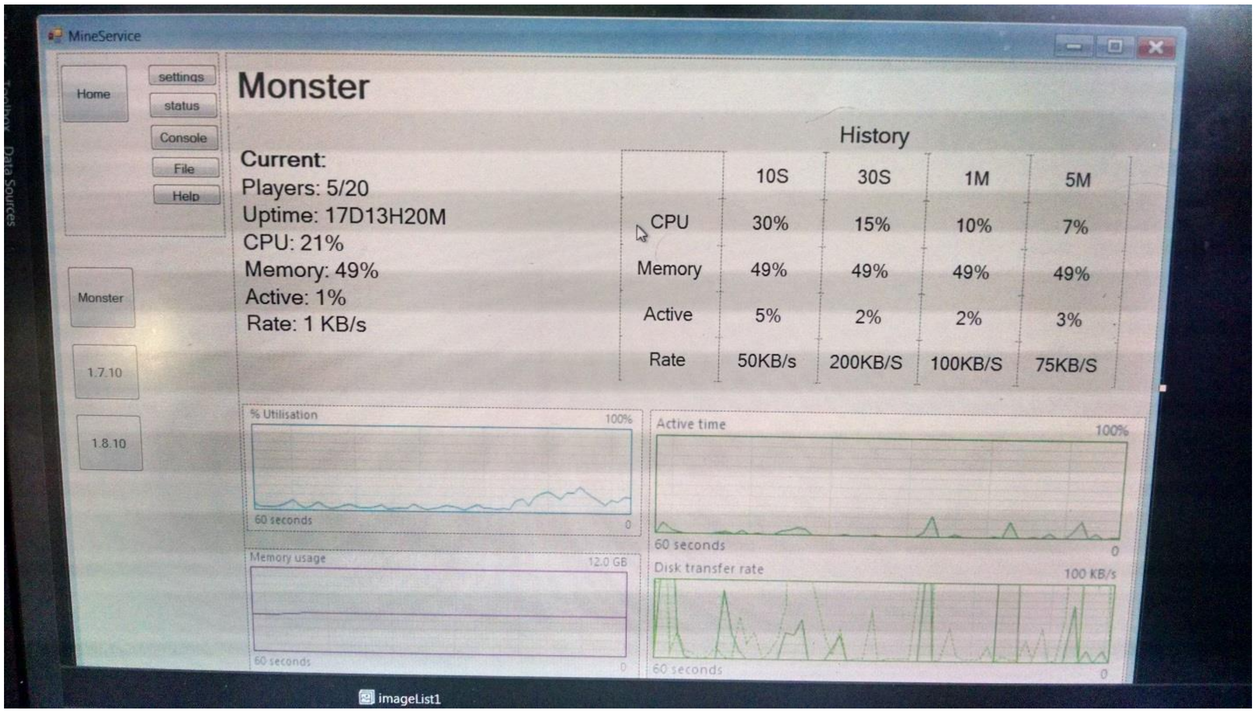
Plan for Evaluation

A couple clients that currently pay CJ were given the prototype designs. The process of them using the prototypes was recorded and put into a video file. We hoped to accomplish the layout of the user interface and obtain feedback on the display of the analytical data of the servers. We were unsure especially of which side the tabs should be on for displaying the other server's data. In addition, we hoped to obtain feedback from the where the commands should be issued.

Result for Evaluation

Based on the feedback, we saw that the "file/help" bar should be removed and create a new tab called "home" that this will integrate in with. The elements under the current overview tab will also be under the "home" tab. Overall the feedback was good, just requested a few things being moved around the UI. They also said the first prototype would be good for the desktop application.

Changes to Prototype



Monster

Current:
Players: 5/20
Uptime: 17D13H20M
CPU: 21%
Memory: 49%
Active: 1%
Rate: 1 KB/s

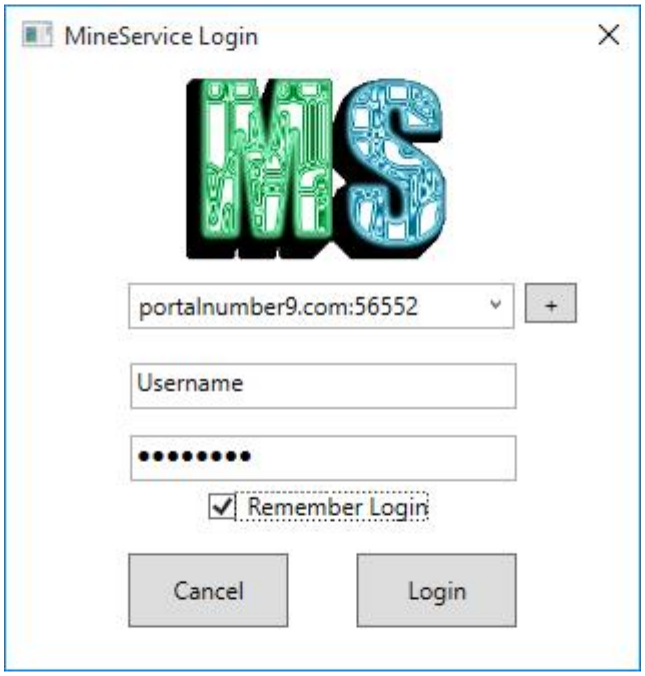
History

	10S	30S	1M	5M
CPU	30%	15%	10%	7%
Memory	49%	49%	49%	49%
Active	5%	2%	2%	3%
Rate	50KB/s	200KB/S	100KB/S	75KB/S

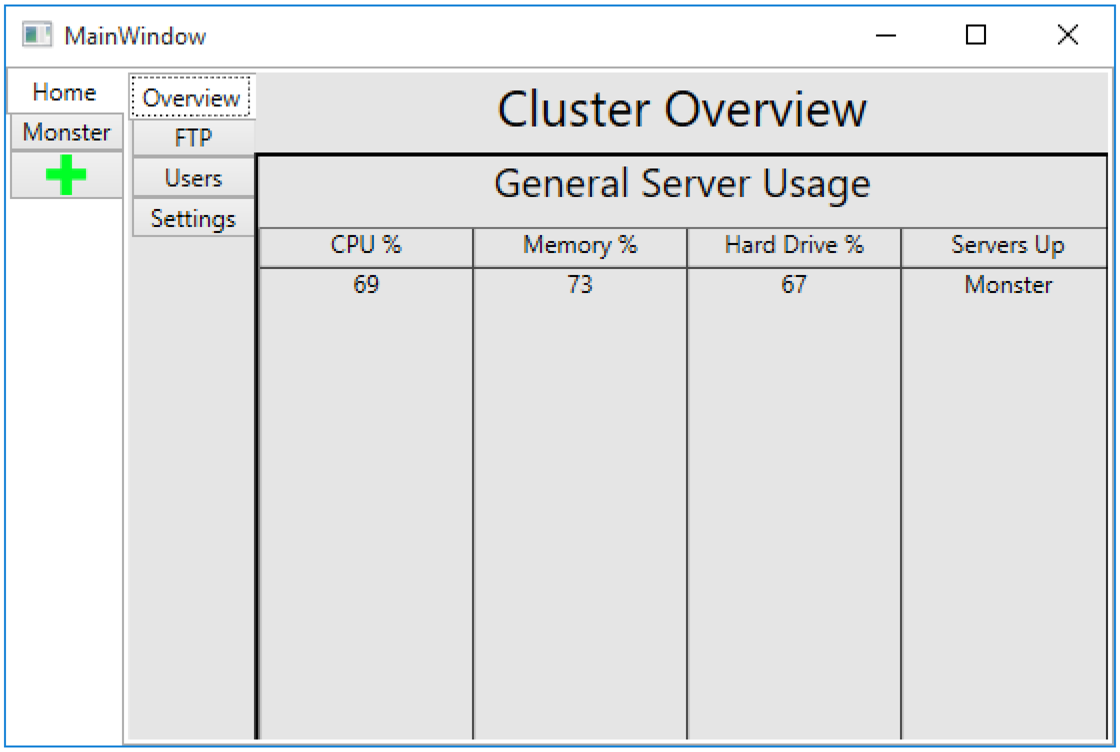
Home tab now involves file and help and the place for entering commands is in the home tab as well.

High-Fidelity Prototype

This is the first window you will see. General Login info that can store the servers you connect to.



This is the window you will see after logging in. General Information about the cluster

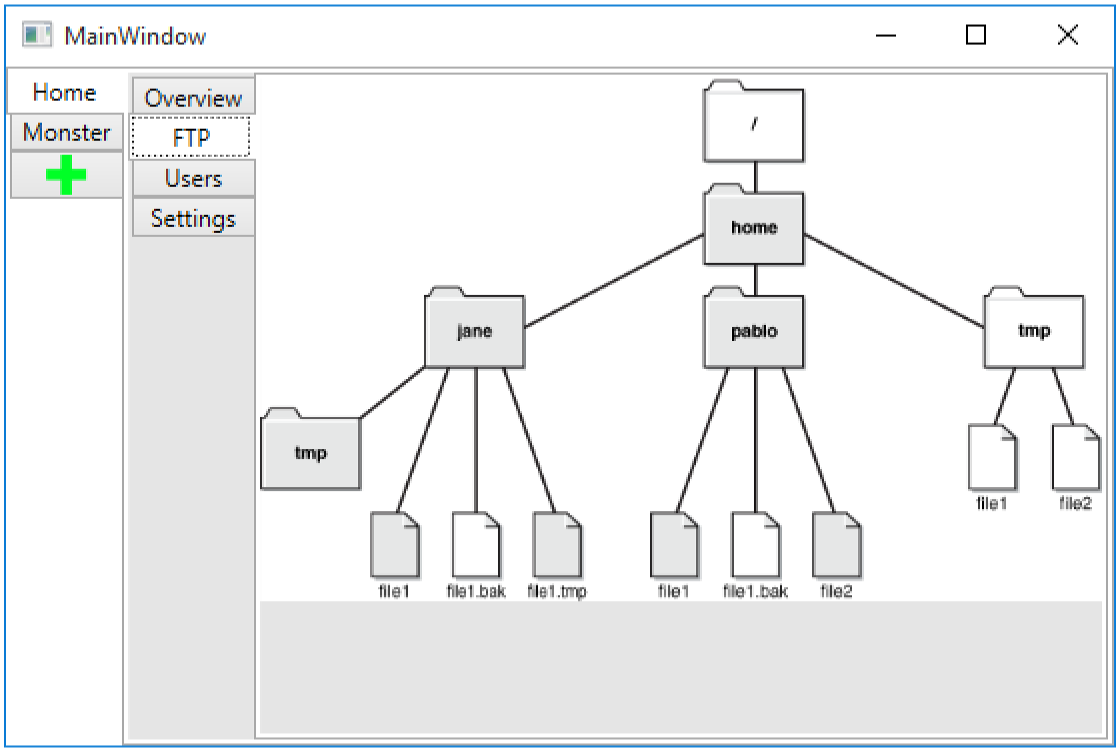


Cluster Overview

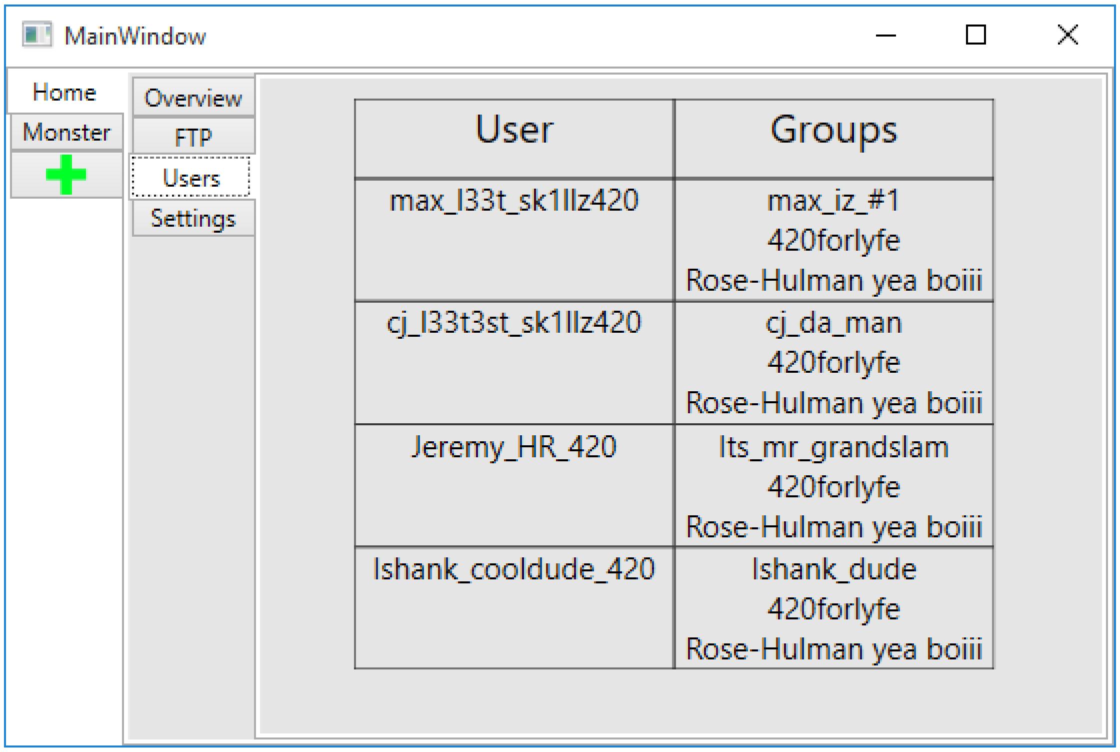
General Server Usage

CPU %	Memory %	Hard Drive %	Servers Up
69	73	67	Monster

This is the FTP: Design still in progress might go to more of a text based solution. Waiting from response from test clients.

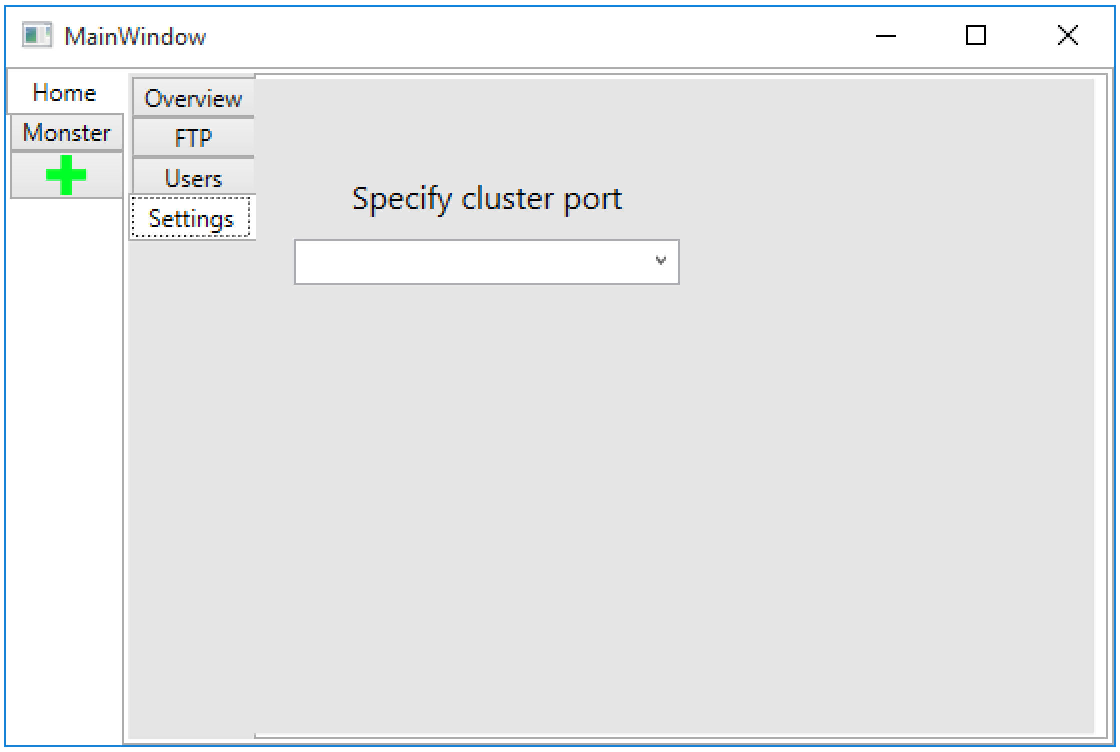


This is basic user control to see who has access to what and in what group.

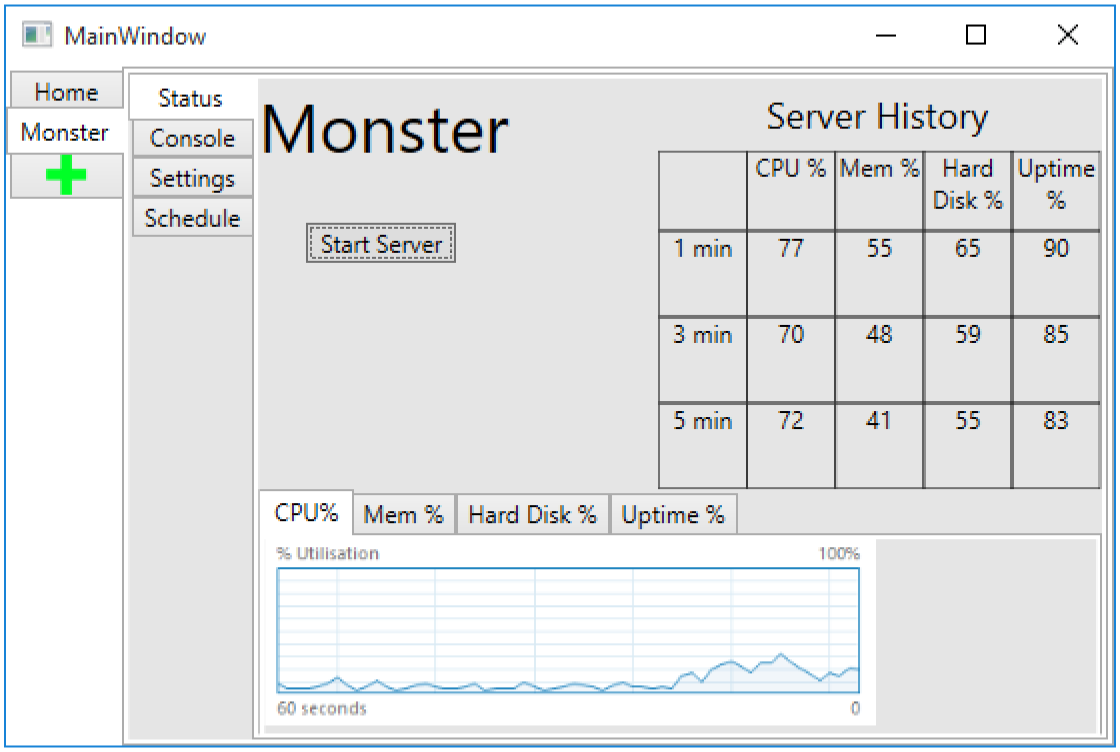


User	Groups
max_l33t_sk1llz420	max_iz_#1 420forlyfe Rose-Hulman yea boiii
cj_l33t3st_sk1llz420	cj_da_man 420forlyfe Rose-Hulman yea boiii
Jeremy_HR_420	Its_mr_grandslam 420forlyfe Rose-Hulman yea boiii
Ishank_cooldude_420	Ishank_dude 420forlyfe Rose-Hulman yea boiii

This page will house cluster settings as they are needed. Currently only port is needed, but latter if we extend to additional features this will be needed.

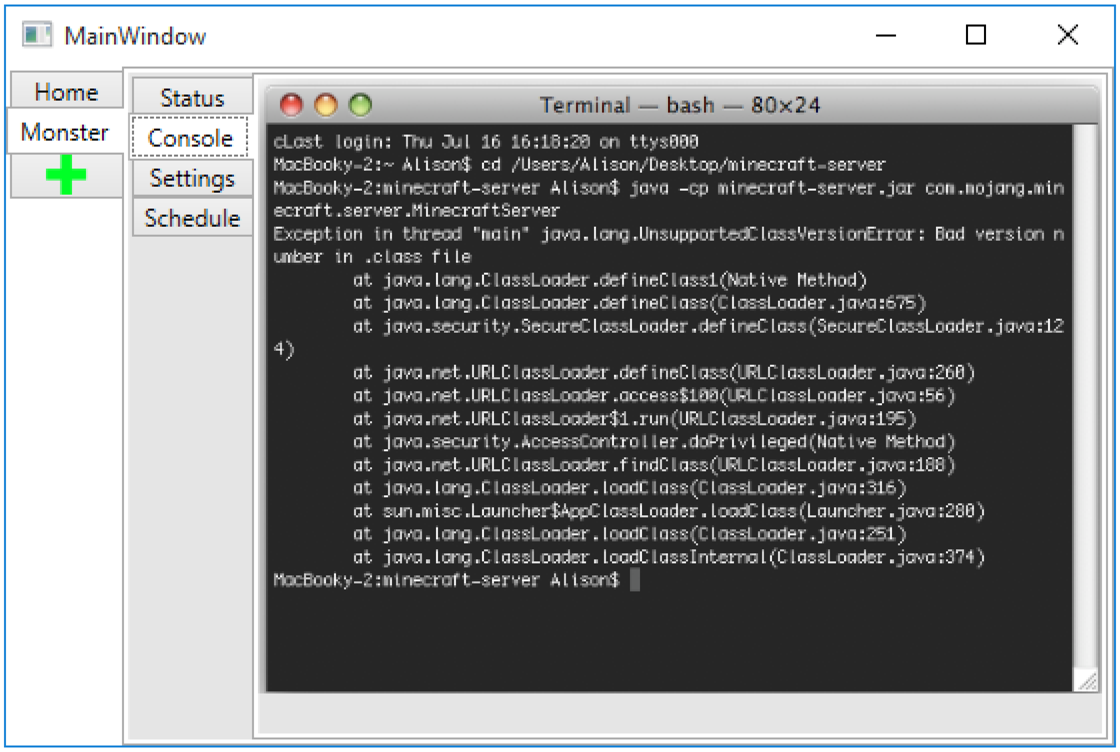


This is the default page for a specific server.



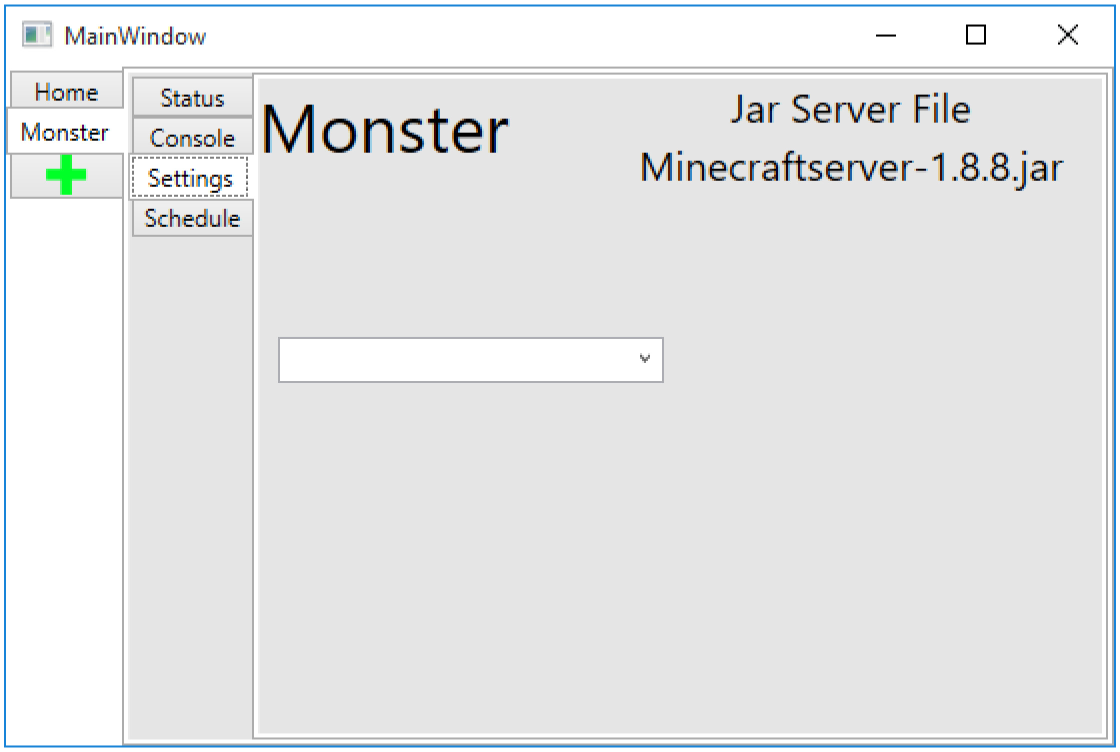
	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

This is the console window for a specific Minecraft server, window still in progress.

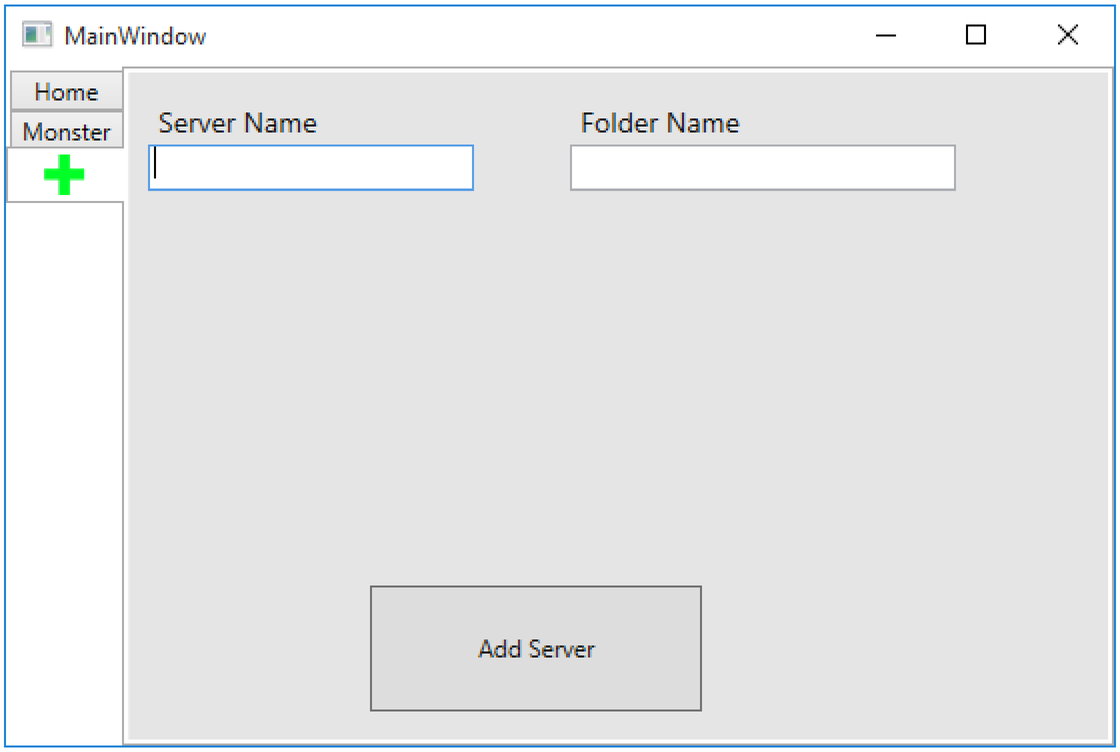


```
cLast login: Thu Jul 16 16:18:28 on ttys000
MacBooky-2:~ Alison$ cd /Users/Alison/Desktop/minecraft-server
MacBooky-2:minecraft-server Alison$ java -cp minecraft-server.jar com.mojang.min
ecraft.server.MinecraftServer
Exception in thread "main" java.lang.UnsupportedClassVersionError: Bad version n
umber in .class file
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:675)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:12
4)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:260)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:56)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:195)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:316)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:280)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:251)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:374)
MacBooky-2:minecraft-server Alison$
```

Server specific settings. Still in progress.



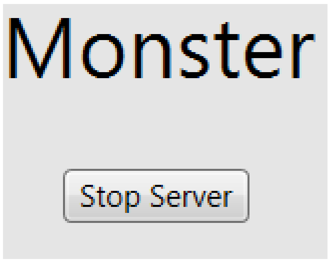
Page to add a new server to the cluster.

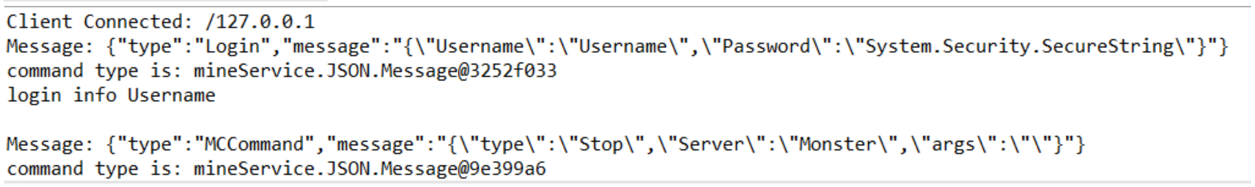


Features Completed

(note: names listed in approximate order of contribution to feature)

Feature: Add a start/stop server button to the GUI, send the correct message to the server, and make sure the server is receiving the message in the correct format.





```
Client Connected: /127.0.0.1
Message: {"type":"Login","message":"{\"Username\":\"Username\",\"Password\":\"System.Security.SecureString\"}"}
command type is: mineService.JSON.Message@3252f033
login info Username

Message: {"type":"MCCommand","message":"{\"type\":\"Stop\",\"Server\":\"Monster\",\"args\":\"\"}"}
command type is: mineService.JSON.Message@9e399a6
```

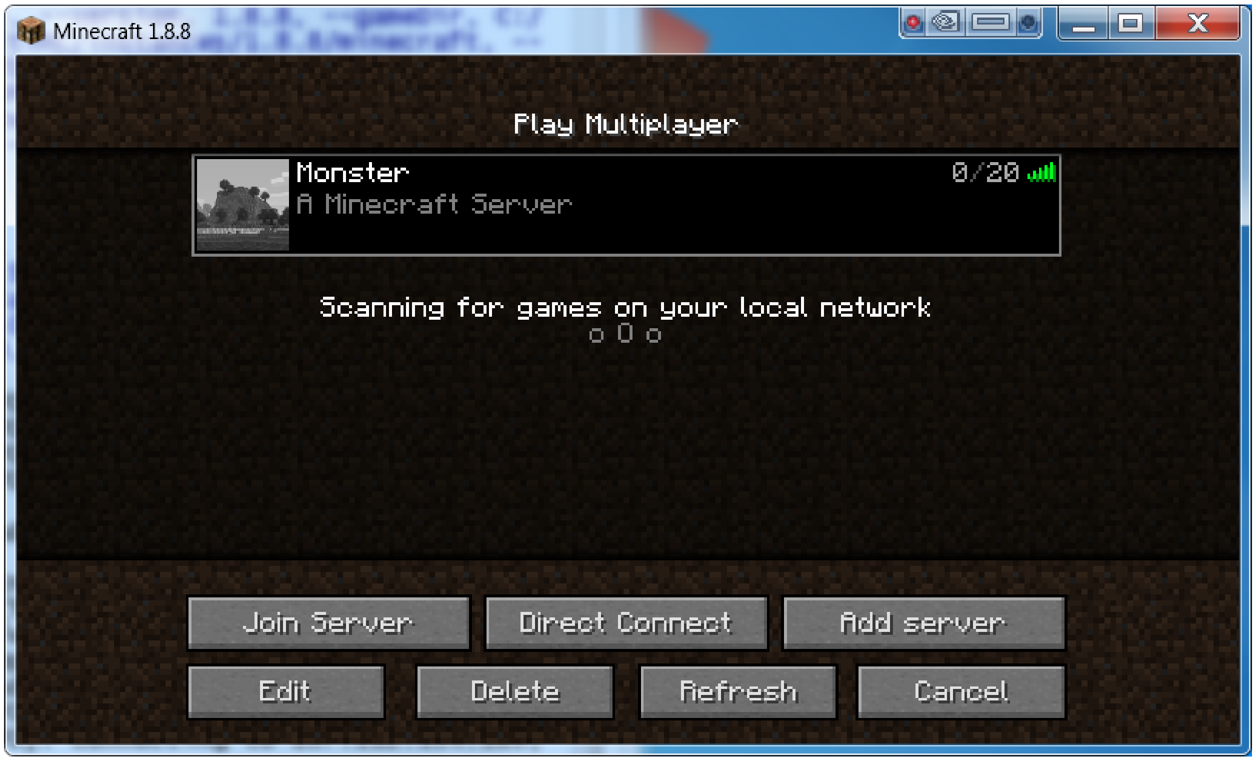
Conditions of Satisfaction

- Acknowledge that I don’t have to use terminal to manage my servers

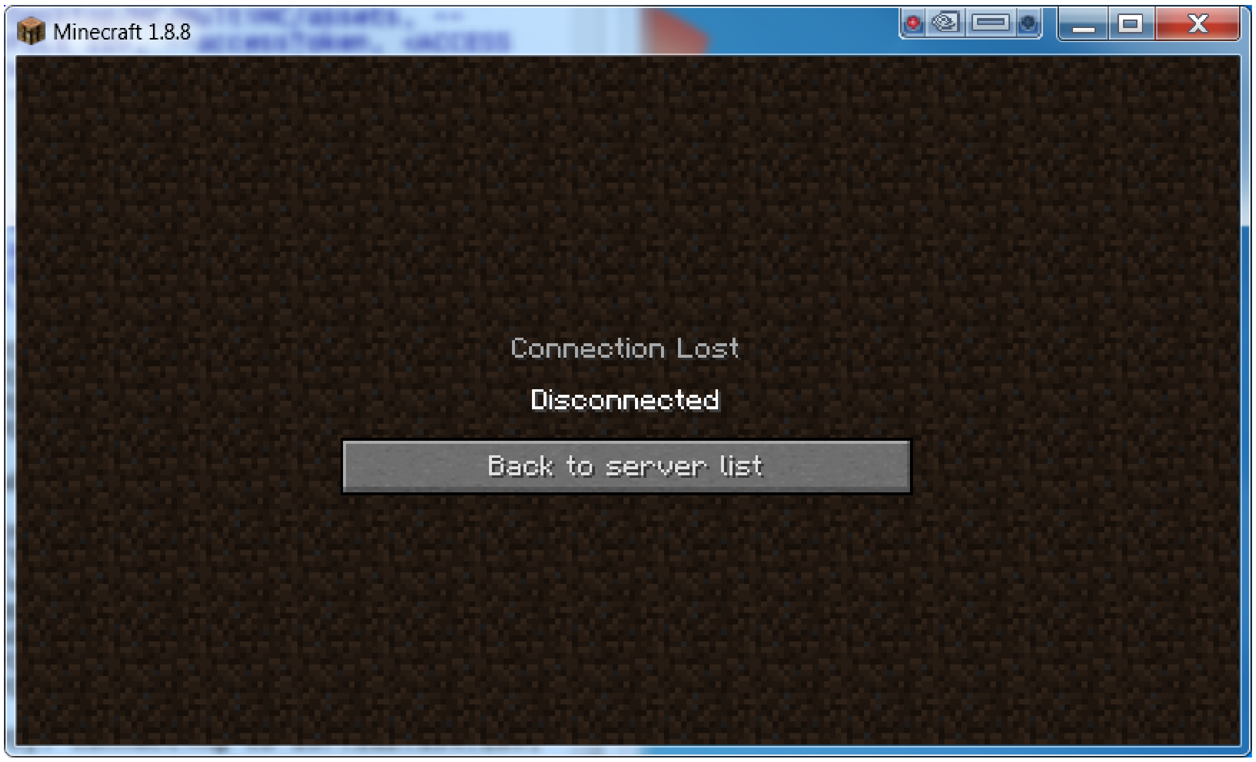
Contributors: Ishank, C.J., Max, Jeremy

Feature: Handle start and stop requests on the server so that a Minecraft server is started or stopped when requested.

Minecraft client a few second after the start message has been received.



After clicking stop on MineService this is seen on the Minecraft client.



On the MineService Server Console the following is seen:

```
{"type":"MCCommand","message":"{\"type\":\"Start\",\"Server\":\"Monster\",\"args\":\"\"}"}
{"type":"MCCommand","message":"{\"type\":\"Stop\",\"Server\":\"Monster\",\"args\":\"\"}"}
```

Conditions of Satisfaction

- When a stop message is received the running server is stopped
- When a start message is received the server is started and only if it is not already running.

Contributors: CJ, Jeremy

Feature: Users are authenticated against a database to be able to login.

Shows on the server when a user logs in with a good username/password:

```
Client Connected: /127.0.0.1
{"type":"Login","message":"{\"Username\":\"Username\",\"Password\":\"System.Security.SecureString\"}"}
login info Username

Login: Success
```

From a bad user/pass combo:

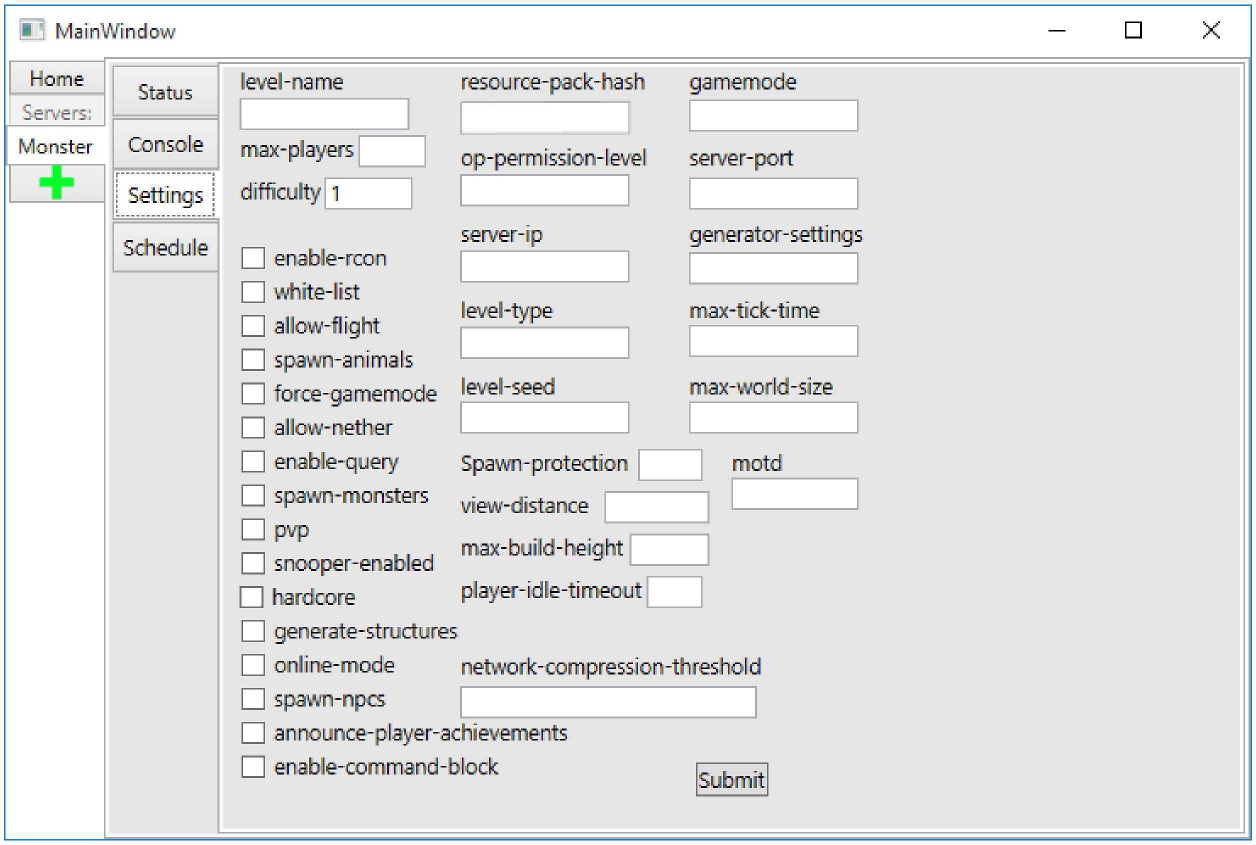
```
Client Connected: /127.0.0.1
{"type":"Login","message":"{\"Username\":\"Username\",\"Password\":\"System.Security.SecureString\"}"}
login info Username

Login: Denied
```

Conditions of Satisfaction

- A login that is stored allows you to connect.
- A login that is not valid disconnects you.

Contributors: Jeremy, CJ

Feature: Improved Settings tab for specific servers with options that represent real possible settings. Also some coding to be used to communicate this information to the server.



level-name
max-players
difficulty 1

- [ ] enable-rcon
- [ ] white-list
- [ ] allow-flight
- [ ] spawn-animals
- [ ] force-gamemode
- [ ] allow-nether
- [ ] enable-query
- [ ] spawn-monsters
- [ ] pvp
- [ ] snooper-enabled
- [ ] hardcore
- [ ] generate-structures
- [ ] online-mode
- [ ] spawn-npcs
- [ ] announce-player-achievements
- [ ] enable-command-block

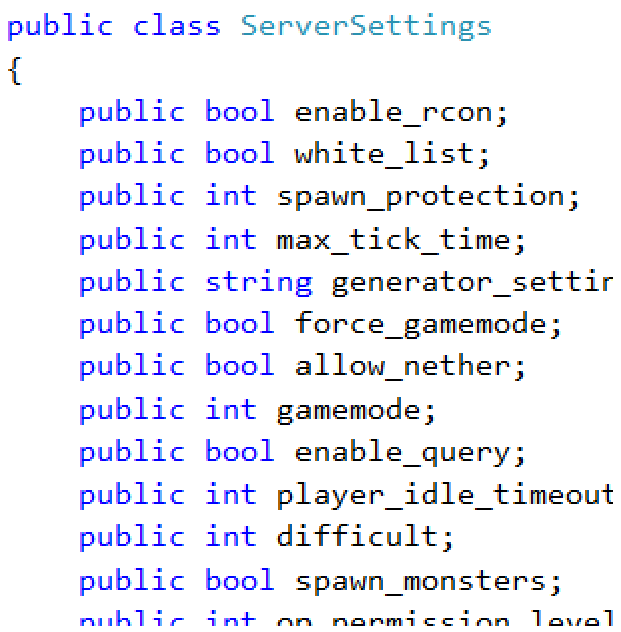
resource-pack-hash
op-permission-level
server-ip
level-type
level-seed
Spawn-protection
view-distance
max-build-height
player-idle-timeout
network-compression-threshold

gamemode
server-port
generator-settings
max-tick-time
max-world-size
motd

The user can enter settings in this window

Conditions of Satisfaction

- The user should be able to perform all the actions they want to perform with the given settings.
- The use of the settings should be intuitive enough for a first-time user to use them correctly.
- The settings should be implemented correctly for the in-game experience.

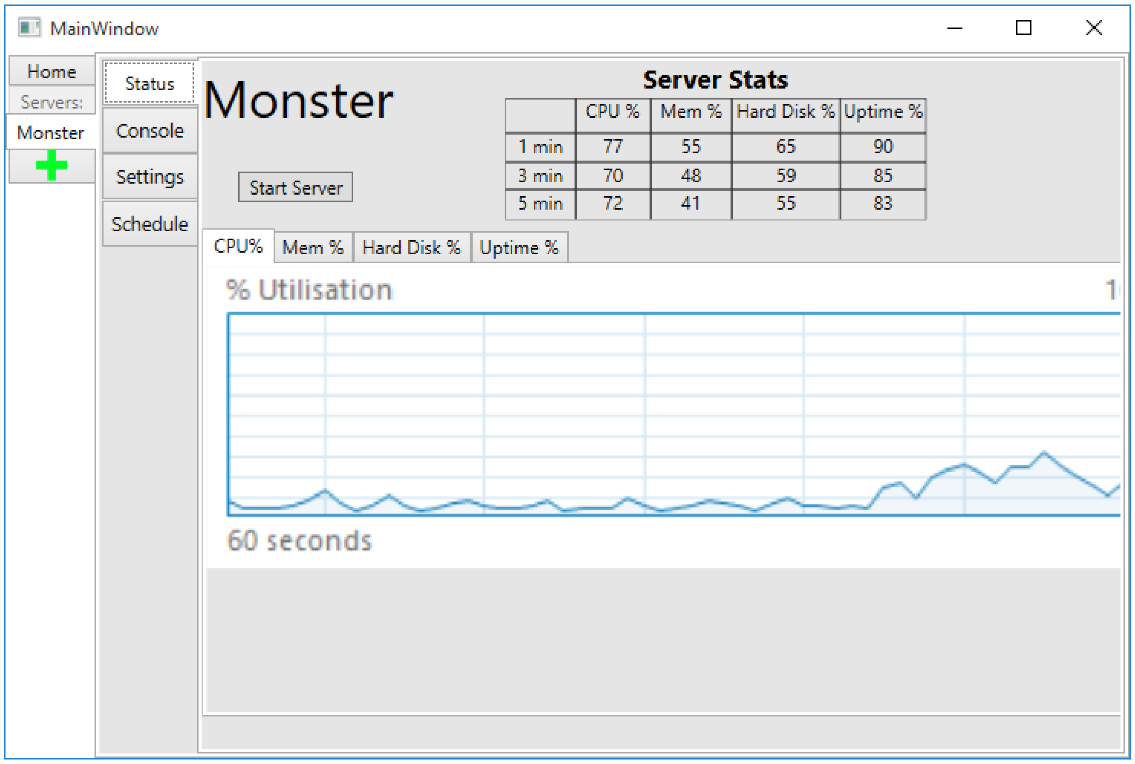


```
public class ServerSettings
{
    public bool enable_rcon;
    public bool white_list;
    public int spawn_protection;
    public int max_tick_time;
    public string generator_settir
    public bool force_gamemode;
    public bool allow_nether;
    public int gamemode;
    public bool enable_query;
    public int player_idle_timeout
    public int difficult;
    public bool spawn_monsters;
    public int op_permission_level
```

This is part of the C# objects that the user can change in the settings window. These objects will be sent to the server where they will change in-game behavior.

Contributors: Max, Ishank, C.J.

Feature: Used Microsoft Blend to improve aesthetics of the application, and resized elements in windows to make better use of space.



Server Stats

	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

Conditions of Satisfaction:

- More appealing and understandable user interface than what existed previously, according to people who didn't design the UI.

Contributors: Max

Business Model Canvas

Key Partners

- Potentially large scale hosting providers

Key Activities

1.) Minecraft server administration

Key Resources

- Development environment

Value Propositions

1.) Centralized tool location for managing MC servers
2.) Provides easy to use interface to improve MC server hosting experience

Customer Relationships

1.) Cost effective
2.) Ability to contact developers in case of an error
3.) Low downtime

Channels

1.) Website
2.) IRC
3.) Forums (External)

Customer Segments

1. Server administrators
2. Minecraft administrators

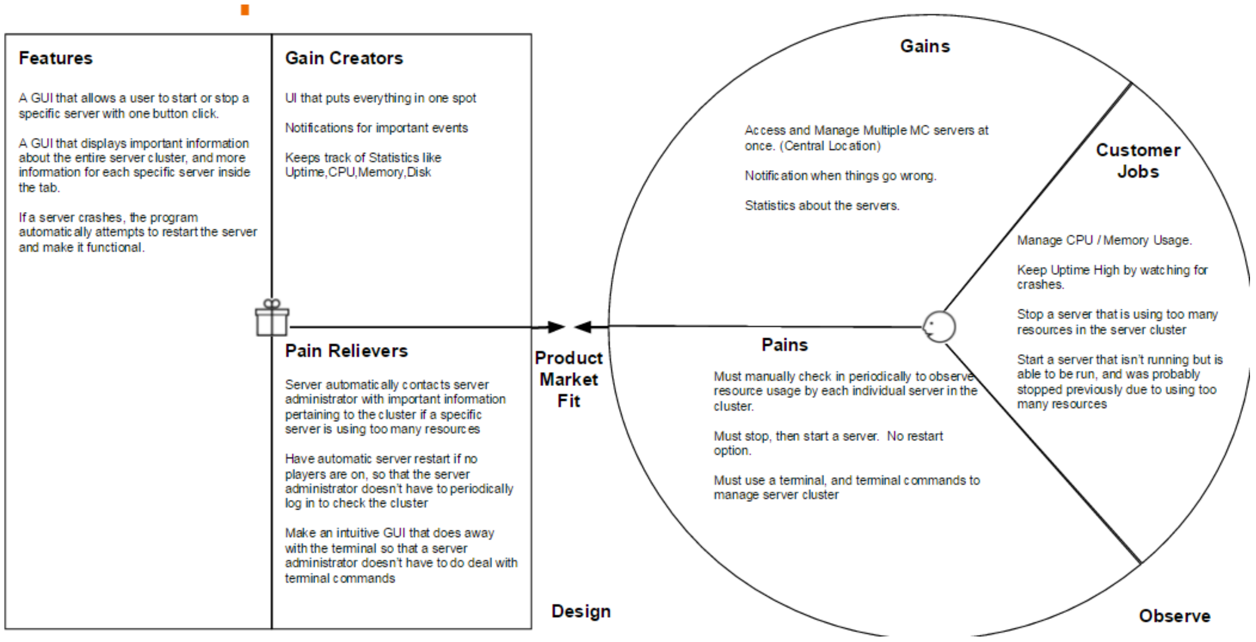
Cost Structure

- salaries
- Website hosting
- Development Tools

Revenue Streams

- Pay for # of servers managed
- Ads on Website

Value Proposition Canvas



Features

A GUI that allows a user to start or stop a specific server with one button click.

A GUI that displays important information about the entire server cluster, and more information for each specific server inside the tab.

If a server crashes, the program automatically attempts to restart the server and make it functional.

Gain Creators

UI that puts everything in one spot

Notifications for important events

Keeps track of Statistics like Uptime,CPU,Memory,Disk

Pain Relievers

Server automatically contacts server administrator with important information pertaining to the cluster if a specific server is using too many resources

Have automatic server restart if no players are on, so that the server administrator doesn't have to periodically log in to check the cluster

Make an intuitive GUI that does away with the terminal so that a server administrator doesn't have to do deal with terminal commands

Gains

Access and Manage Multiple MC servers at once. (Central Location)

Notification when things go wrong.

Statistics about the servers.

Customer Jobs

Manage CPU / Memory Usage.

Keep Uptime High by watching for crashes.

Stop a server that is using too many resources in the server cluster

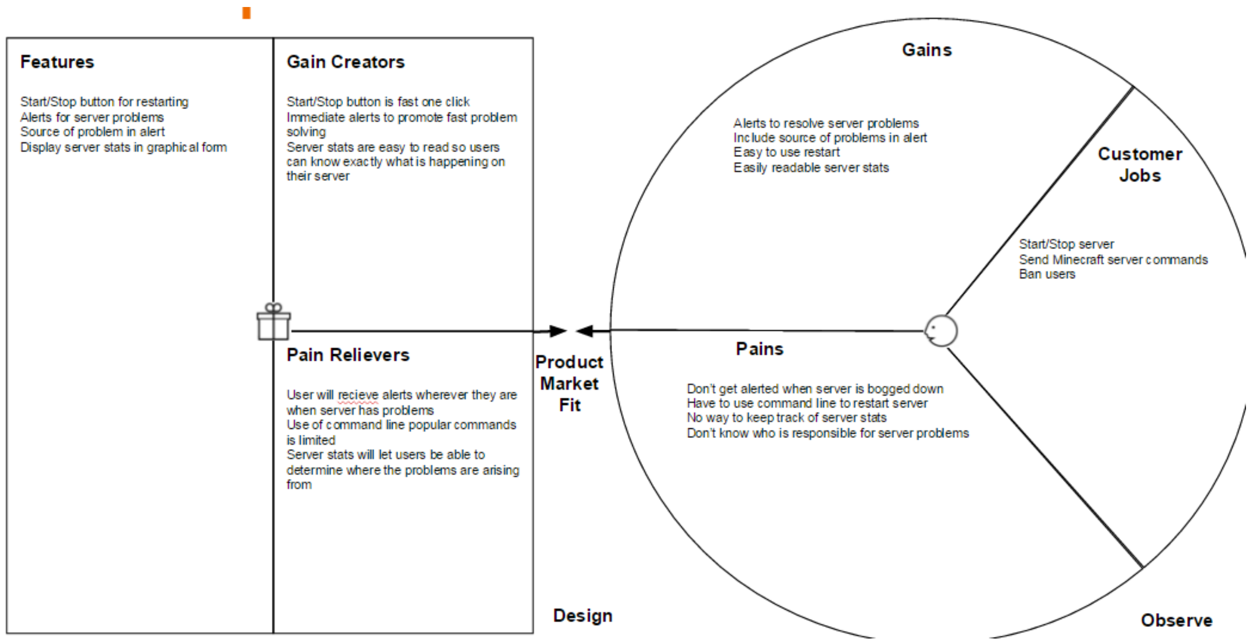
Start a server that isn't running but is able to be run, and was probably stopped previously due to using too many resources

Pains

Must manually check in periodically to observe resource usage by each individual server in the cluster.

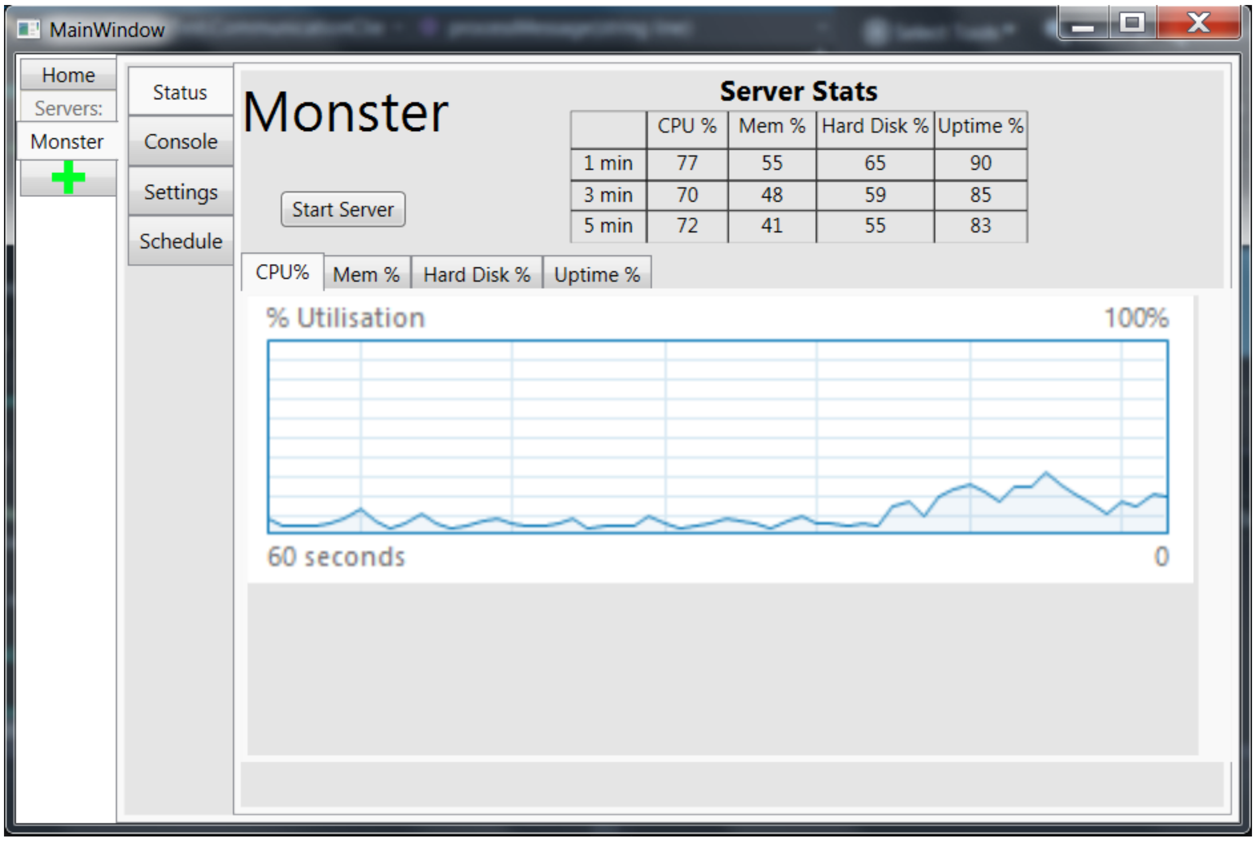
Must stop, then start a server. No restart option.

Must use a terminal, and terminal commands to manage server cluster


Features
Start/Stop button for restarting
Alerts for server problems
Source of problem in alert
Display server stats in graphical form
Gain Creators
Start/Stop button is fast one click
Immediate alerts to promote fast problem solving
Server stats are easy to read so users can know exactly what is happening on their server
Pain Relievers
User will recieve alerts wherever they are when server has problems
Use of command line popular commands is limited
Server stats will let users be able to determine where the problems are arising from
Product Market Fit
Design
Gains
Alerts to resolve server problems
Include source of problems in alert
Easy to use restart
Easily readable server stats
Customer Jobs
Start/Stop server
Send Minecraft server commands
Ban users
Pains
Don't get alerted when server is bogged down
Have to use command line to restart server
No way to keep track of server stats
Don't know who is responsible for server problems
Observe

Usability Study Improvements

The feedback from the usability lab was pretty positive and not much had to be changed. The main problem we saw was that clients were having trouble finding the individual server pages, which we plan to improve upon by making a "Servers" heading and placing the servers under that heading. This change is reflected on the left hand side of the application window in the following screenshot.



Server Stats

	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

The usability report can be accessed on the usability lab computer in the CSSE 371 class, 1516, section 2, and in the MineService folder.

CSSE371 Appendix

- Items available upon request:
    - Interview documents
    - Usage Diagrams
    - Initial Concept Pictures and Notes
    - Notes Taken on User usage

CSSE375 Milestone 2

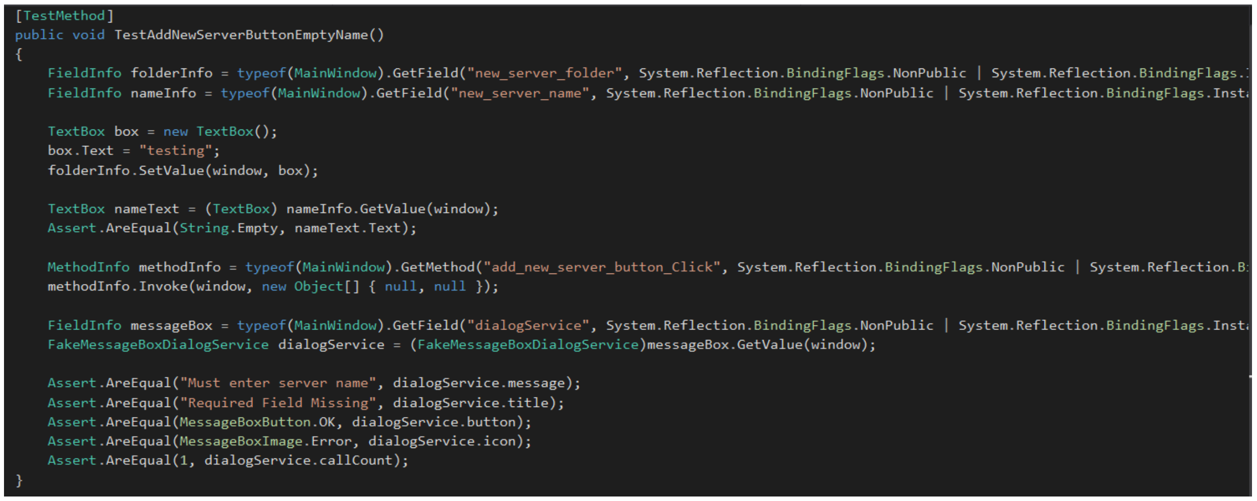
1. The main feature that our team would like to complete for Milestone 2 is the implementation of encryption for the both the client and server as a part of our message passing service. This is the last feature that was mentioned in the project plan from CSSE371. With this feature, the MineService will be more secure and make it harder for unsuspecting parties to fetch data over the wire. The Trello backlog has this displayed, as well as a good list of possible quick-win implementations for the application. Our main focus this week is on encryption, but if time allows we will start to chew away at this existing backlog. In terms of priority, we plan to implement showing memory, CPU, and disk usage to the client for Milestone 3, with the extended goal to get those as lifetime graphs.
2. In order to add our encryption feature to the program functionality, we needed to add an IMessageControl interface that allows for reading and writing of a stream object. This was a slight addition that needed to happen for encryption to be implemented.
3. The team has gone through the code and identified quite a few bad code smells that we have found in the code base. The list of smells found are posted in the Trello Main Board in the Refactor Backlog lane, and are also provided in more detail below:
    A. The add_new_server_button_Click function in the MineService.MainWindow.xaml.cs file. This function is called whenever a client attempts to add a new server, and handles errors if the server folder or name is empty. The two bad smells would be both duplicate code and speculative generality. The if statements can be restructured, and the String checks can be made into variables for easier access (*duplicate code*). The *speculative generality* comes from the Message being sent to the server. This method is somewhat doing what it is not titled to do; I do not see why this button click function handles sending a message to a server. The extract method refactoring idea would work well here to make the method more coherent and the code more readable. **(*Collin will refactor this smell for the next Milestone)***
    B. The processMessage function in MineService.CommunicationClient.cs is another prime example of a bad code smell that we found. This class has a bad smell with a lot of code inside of a switch statement where each case could be extracted into a method. The conversion from json to string will be refactored with a different bad code smell that Brandon is working on. The combination of these fixes will make the processMessage method more clear and concise. The switch statement will get help from the new methods so that it is more readable to see what the processMessage method is doing. **(*Collin will refactor this smell for the next Milestone)***
    C. The UpdateTab method in MineService.ServerTabItem.xaml.cs has many if statements that could be considered poor control structure and duplicate code. Although this is not a focus for any team member for next milestone, this code

could easily be refactored by thinking about the control structure more thoroughly.

D. Another code smell we have identified that will remain untouched for now is the fact that in the MineService Client JSON project, all JSON objects act similar but do not inherit common behavior from an interface or superclass. Once Brandon puts the convertToJSON functions in these classes, it is necessary that they all implement a common MSJSON interface with a convertToJson method. This makes it easier for maintenance if more methods need to be added to JSON objects. This will also allow someone new on the team to add a JSON object easily with implementing proper behavior.
E. MCServer should be abstracted so that new types of servers can be added. This can be acomplished by making an interface that has the methods and then also making an abstract class that does default implementation of methods. This will allow adding diffrent types of servers such as Modded Minecraft servers or even diffrent programs entirely without having to change any code. **(*CJ will refactor this smell for the next Milestone)***
F. The method has several smells. The most pronounced is long method which is a result of duplicate code. There also also nested classes inside the method that can be extracted. After extracting these they can be referenced in the original method and use aggregation / delegation for each of these. Making these abtracted classes invoke an interfance that is related to what they do so that the refactor overall for MCServer can be abstracted. **(*CJ will refactor this smell for the next Milestone)***
G. The startProcessing method on the server is complicated and has a private internal class. This is something we are not refactoring for the next milestone, but could definitely consider a restructure and extract methods for this function.
H. The processMessage method on the server side has a switch statement that needs refactored. The handler methods that processMessage calls to could also be made more general and use some extract method refactoring.
I. In MineService MainWindow.xaml.cs getData function. When looking at this method, we can easily identify a problem when dealing with the switch statement that makes up most of the method. Although we may not use this switch statement elsewhere in the code, the idea of polymorphism here would be beneficial to us, or future programmers, if they need to add other TabItems. Taking this switch statement and moving it to its class will allow for easier changes to be made if other TabItems are introduced into the system. **(*Brandon will refactor this smell for the next Milestone)***
J. JsonConvert Duplicate Code. When using JsonConvert, we find that we call this and some methods it evokes many times during the course of running the program as we need it to help with messages between the server and the client. We can extract the method, create a new call to it, and replace the current calls to the new

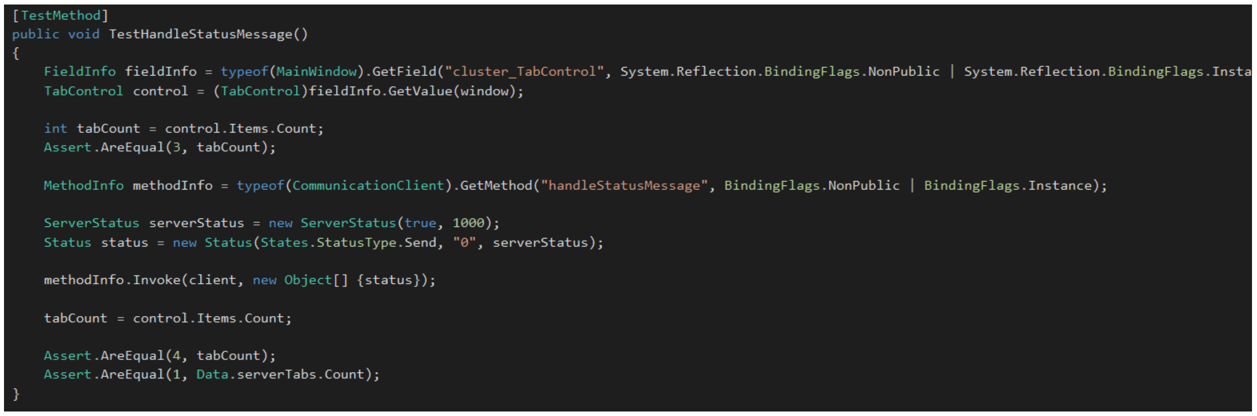
method. This will help with the duplicate code issue as we will only need to make changes in one spot to adjust the JsonConvert rather than going to each instance within the code itself. ***(Brandon will refactor this smell for the next Milestone)***

4. **The test cases for the 6 bad code smells refactors are listed below:**
   TestAddServerButtonClick from MainWindow.xaml.cs.



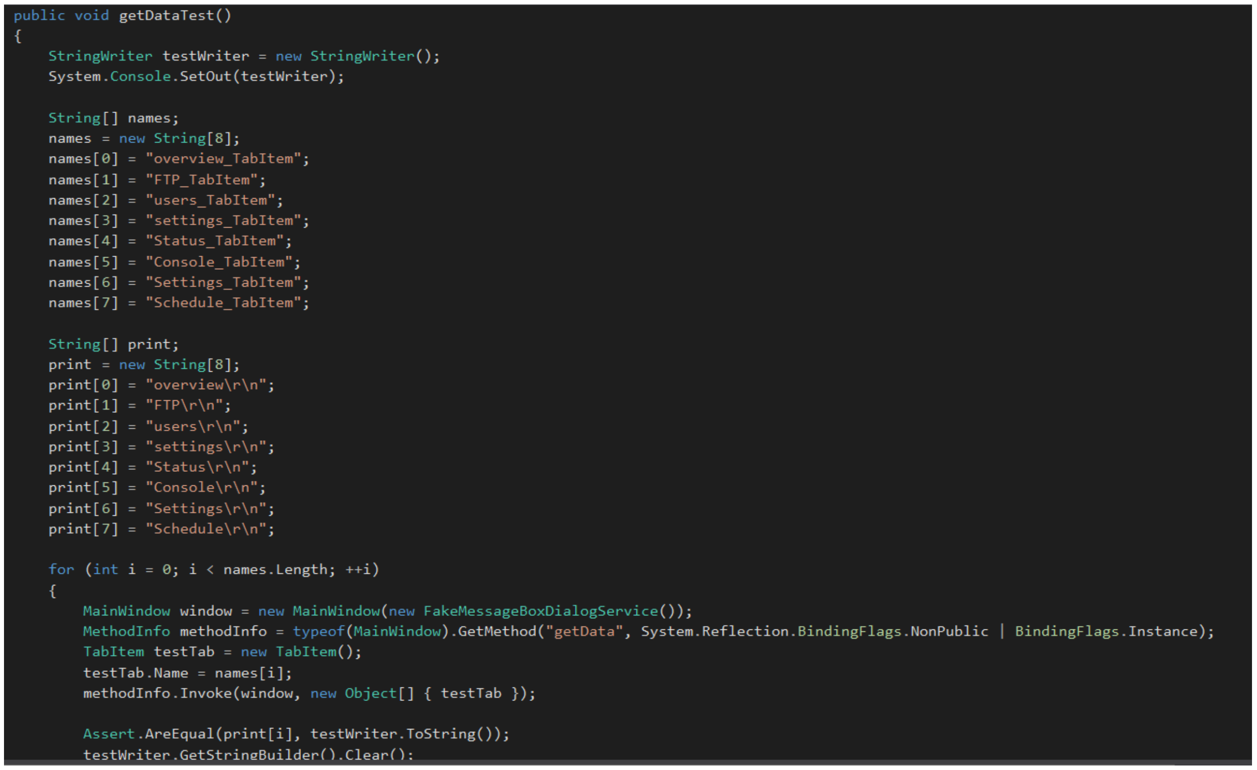
```
[TestMethod]
public void TestAddNewServerButtonEmptyName()
{
    FieldInfo folderInfo = typeof(MainWindow).GetField("new_server_folder", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.
    FieldInfo nameInfo = typeof(MainWindow).GetField("new_server_name", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Insta

    TextBox box = new TextBox();
    box.Text = "testing";
    folderInfo.SetValue(window, box);

    TextBox nameText = (TextBox) nameInfo.GetValue(window);
    Assert.AreEqual(String.Empty, nameText.Text);

    MethodInfo methodInfo = typeof(MainWindow).GetMethod("add_new_server_button_Click", System.Reflection.BindingFlags.NonPublic | System.Reflection.B
    methodInfo.Invoke(window, new Object[] { null, null });

    FieldInfo messageBox = typeof(MainWindow).GetField("dialogService", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Insta
    FakeMessageBoxDialogService dialogService = (FakeMessageBoxDialogService)messageBox.GetValue(window);

    Assert.AreEqual("Must enter server name", dialogService.message);
    Assert.AreEqual("Required Field Missing", dialogService.title);
    Assert.AreEqual(MessageBoxButton.OK, dialogService.button);
    Assert.AreEqual(MessageBoxImage.Error, dialogService.icon);
    Assert.AreEqual(1, dialogService.callCount);
}
```

   Test HandleStatusMessage. This was extracted from processLine in the communication client.



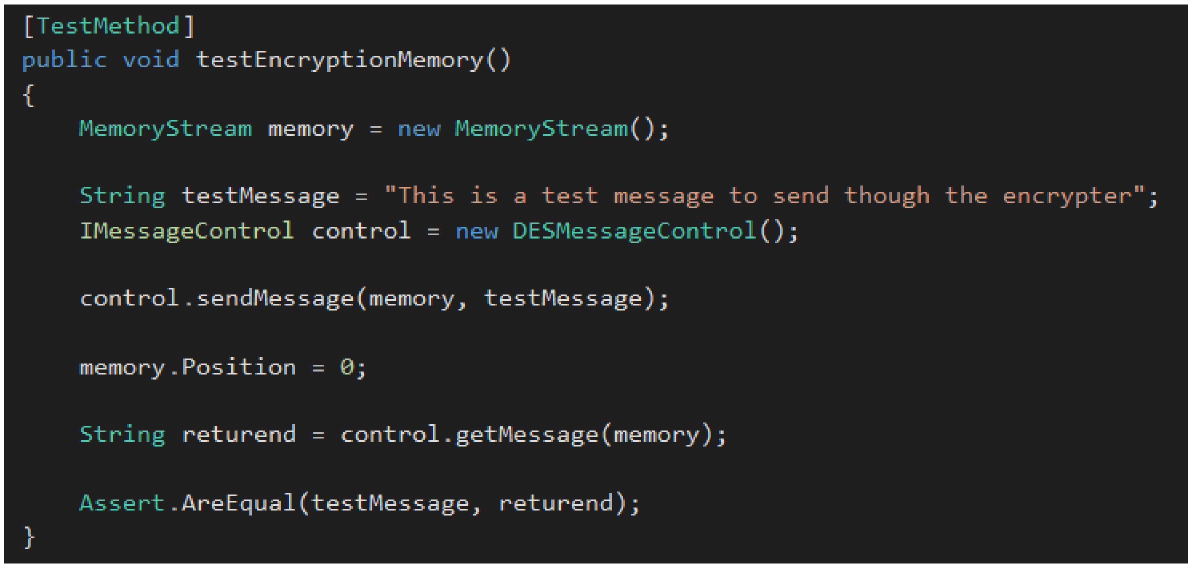
```
[TestMethod]
public void TestHandleStatusMessage()
{
    FieldInfo fieldInfo = typeof(MainWindow).GetField("cluster_TabControl", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Insta
    TabControl control = (TabControl)fieldInfo.GetValue(window);

    int tabCount = control.Items.Count;
    Assert.AreEqual(3, tabCount);

    MethodInfo methodInfo = typeof(CommunicationClient).GetMethod("handleStatusMessage", BindingFlags.NonPublic | BindingFlags.Instance);

    ServerStatus serverStatus = new ServerStatus(true, 1000);
    Status status = new Status(States.StatusType.Send, "0", serverStatus);

    methodInfo.Invoke(client, new Object[] {status});

    tabCount = control.Items.Count;

    Assert.AreEqual(4, tabCount);
    Assert.AreEqual(1, Data.serverTabs.Count);
}
```

Test GetData.



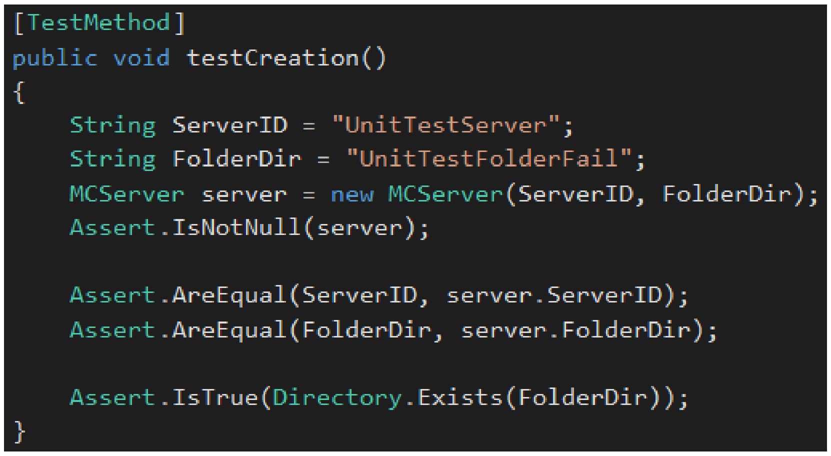
```
public void getDataTest()
{
    StringWriter testWriter = new StringWriter();
    System.Console.SetOut(testWriter);

    String[] names;
    names = new String[8];
    names[0] = "overview_TabItem";
    names[1] = "FTP_TabItem";
    names[2] = "users_TabItem";
    names[3] = "settings_TabItem";
    names[4] = "Status_TabItem";
    names[5] = "Console_TabItem";
    names[6] = "Settings_TabItem";
    names[7] = "Schedule_TabItem";

    String[] print;
    print = new String[8];
    print[0] = "overview\r\n";
    print[1] = "FTP\r\n";
    print[2] = "users\r\n";
    print[3] = "settings\r\n";
    print[4] = "Status\r\n";
    print[5] = "Console\r\n";
    print[6] = "Settings\r\n";
    print[7] = "Schedule\r\n";

    for (int i = 0; i < names.Length; ++i)
    {
        MainWindow window = new MainWindow(new FakeMessageBoxDialogService());
        MethodInfo methodInfo = typeof(MainWindow).GetMethod("getData", System.Reflection.BindingFlags.NonPublic | BindingFlags.Instance);
        TabItem testTab = new TabItem();
        testTab.Name = names[i];
        methodInfo.Invoke(window, new Object[] { testTab });

        Assert.AreEqual(print[i], testWriter.ToString());
        testWriter.GetStringBuilder().Clear();
```

Encryption-Test



```
[TestMethod]
public void testEncryptionMemory()
{
    MemoryStream memory = new MemoryStream();

    String testMessage = "This is a test message to send though the encrypter";
    IMessageControl control = new DESMessageControl();

    control.sendMessage(memory, testMessage);

    memory.Position = 0;

    String returend = control.getMessage(memory);

    Assert.AreEqual(testMessage, returend);
}
```

ServerCreation-Test



```
[TestMethod]
public void testCreation()
{
    String ServerID = "UnitTestServer";
    String FolderDir = "UnitTestFolderFail";
    MCServer server = new MCServer(ServerID, FolderDir);
    Assert.IsNotNull(server);

    Assert.AreEqual(ServerID, server.ServerID);
    Assert.AreEqual(FolderDir, server.FolderDir);

    Assert.IsTrue(Directory.Exists(FolderDir));
}
```

CSSE375 Milestone 3

The bad smell refactorings that we identified in Milestone 2 were tested for and then refactored in this milestone. What follows is a discussion about each of our individual refactorings.

Collin

Add_new_server_button_Click in MainWindow.xaml.cs

I removed the nasty if statements and replaced it with a switch. This switch statement switches on a call to a helper method that I extracted called getCreateServerError. This method checks both text fields for null and then does a bitwise shift, followed by a bitwise or to return either a 0,1,2 or 3. The switch statement then uses those integers as cases, and sets the appropriate message to be displayed to the client. If both fields are not null (case 0), I extracted a method called createServer which handles the creation of a new server.

processMessage in CommunicationClient

This code was refactored using an extract class refactor. An IMessageHandler was introduced as well as a concrete MessageHandler. The CommunicationClient now composes an IMessageHandler, easily allowing us to fake the handler during testing. The interface contracts a public handleMessage function which converts the message into an object that we check the type for and call the appropriate method. These method calls have been extracted into their own method instead of just being inside of the handleMessage method.

CJ

MCServer

.Net's Process management has built in events. These events corresponded to the same things that there were subclasses before in start server are now are just event handlers. This made this a simple and fast change.

MCServer, AbstractServer, IServer

MCServer now has a type structure with an Interface and an abstract class. I moved all methods that did not have anything Minecraft specific up to AbstractServer. From there I realized that start could be split into two methods. By doing this I could pull up start to the abstract server and the other method became the one that had the minecraft specific parts. I marked that method abstract in AbstractServer.

Brandon

getData in MainWindow.xaml.cs

Originally, the getData method within the MainWindow class on the client side of the program had no true implementation. It used a switch statement to just return the name of the message type to the user. What I did was to add the functionality to this method. The program will pass a tab item, such as the overview tab, and send a message to the server to return the requested information to the client. I was also able to remove the switch statement within the method by using the Factory Pattern and being able to identify which tab item is passed using it.

Shared/JSON

The next refactor was to move the JSON conversion logic into its own object and to extract the respective methods throughout the program where they are used. This helps to remove many instances of duplicate code within the program. It also allows for easy modification to the JSON conversion logic as all the code relating to it is in one place, rather than having to go through all the code to every instance and add the given changes.

Additional Refactors (One per team member)

As part of milestone 3, we also had to go over the code once again and select one additional refactor for team member. These refactorings are individually discussed below.

Collin

UpdateTab in serverTabItem.xaml.cs

The if statements in the updateTab function were refactored and made a little more clear for someone who is new to the code. I removed the outer if statement and changed the order of the checks, enabling a simple else statement to be written to assign the aliveTimeText.Text = "Offline". Tests were written before the refactor and passed after the refactor, so I was sure to not break this logic.

CJ

Shared/JSON

Made json object inherit from the common superclass of Message. This change broke a few files which I fixed due to duplicate variable names. After I made them all have a common superclass I then removed the type field from message. This broke many files (20+). The fix to each of this was simple from changing "msg.type = " to "msg is type". As apart of this I also made message have toJson and fromJson methods which were used in the same spots as the other errors. This allowed me to remove the reference to newtonsoft from the client and server. Also each use of message became a one line instead of wrapping a message inside a message.

Brandon

Server/Client_ProcessMessage

The last refactor was to refactor the switch statement in Client_ProcessMessage. Within the switch statement, we had different code being run for each case which made the method very long and hard to understand. So, to make it easier to work with, I extracted each case and made each their own method that is called when the case is run. This makes the switch statement easier to understand and keeps everything in order.

CSSE375 Milestone 4

Refactorings

Brandon

MCServer – getServerSettings

This method was very long and complex. For this refactoring, I pulled out the parsing logic into its own method(s) to help simplify the steps in the translation from the Java Properties to our JSON object. It also helped to separate the various properties out by type to help keep things organized and ordered.

MCServerTest – testGetServerSettings

The test was implemented to ensure that when the properties are first generated that they resolve to their default values. So this test ends up generating a new MCServerSettings object and checks all the various settings to ensure they are generated to their default values. This helped to make sure that the core functionality of getting the server settings was not altered during the refactor.

Collin

Client_ProcessMessage – create

The create server method inside Client_ProcessMessage was smelling of complex logic and duplicate code. This was hard to understand as a developer, and needed to have an extract method and logic rearrangement as a refactor. The checking of the two text fields was extracted into a method that does bit manipulation to return an error code. The code is used on a switch statement and handled appropriately. The code in the extracted method is commented to ensure future developers know what is going on.

Client_ProcessMessage – tests

Tests were implemented before performing the said refactor in Client_ProcessMessageTest. Reflection was used in order to execute the create method inside Client_ProcessMessage. The FakeClient class was created and used to get the message that was sent back to the client. Assertions were used to ensure that the expected message was sent on to the client.

CJ

Server.Config

To change the Server Config over to be a type of message reduced the size of code in Server->Config. All references to seralizable could be removed and where there was IO references it was changed to Message class. There were two new methods in Message class also to allow loading from File with a given filename and also to save to it.

Test Plan and Approach

For each refactoring opportunity that was executed, we wrote tests before changing the internal structure of the software. We have a testing project called Testing with test cases that correspond to the refactorings. In order to make the system a little less dependent, we have had to introduce two fake classes. The FakeMessageBoxDialogService and FakeMessageControl classes mock out some of the system functionality, so that we can isolate our code for unit testing. This approach has helped us test the system and write tests that are specific to behavior that needs verified.

We currently have 27 passing tests, and these cover a range of functionality throughout our product. We have tests around the new refactoring, as well as for recent features such as encryption. Our approach was to footprint our test cases around refactoring opportunities before they were tackled. Once we could ensure our system was working before the desired change, these tests offered something to regress upon post-refactoring. Not only did tests provide a benefit for refactoring, they also allowed us to know if new functionality had the broken existing system.

Test Suite Summary

TestClass

MainWindowTest.cs

- *getData* – this test ensures correct json string is fired for the corresponding server tab item.
- *testAddNewServerButton** – these three tests test different cases of how server buttons are created. They ensure proper fields are filled and the correct error is sent back.
- *testCreateNewServer* – ensures message is sent to the server when a new server is requested to be created.
- *testGetServerErrorCode** – these tests make sure the bitwise operations return the proper values and use reflection to access the private method getCreateServerError.

MessageHandlerTest.cs

- *handleStatusMessage** – makes sure when the client receives a new server status it properly populates a new tab.
- *testHandleNewWindow* – this is a more abstract of the tests above. **Warning:** this test may return inconclusive as this is a multi-threading test. Execute manual test if this test returns inconclusive.

ServerTabItemTest.cs

- *All tests* – All of the tests in this class make sure the server up time and online status is properly displayed to the client.

Client_ProcessMessageTest.cs

- *TestCreateServer** - these tests pass in field values for the server name and folder and ensure the proper error message gets sent back to the client.

MCServerTest.cs

- *startThreadInitilizationTest** - These two tests make sure that when a server is started that it actually starts or throws the proper error message.
- *testDelete** - these tests check that when delete is called, the server is deleted and or files are deleted on the file system.
- *testGetServerSettings* – this checks that the server returns a correct status object with the expected properties.
- *testGetBaseStatus* – this checks that the server always returns a status object.
- *testCreation* – this checks that when a new server is created, that its' file system is also initialized.

DESEncryptionTest

- *All tests* – These tests check to make sure that messages are properly handled such that they are encrypted and decrypted correctly both locally and across the network.

Mock_Fake

- This folder contains all of our mocked and fake classes used in the Testing project.

TestFolder

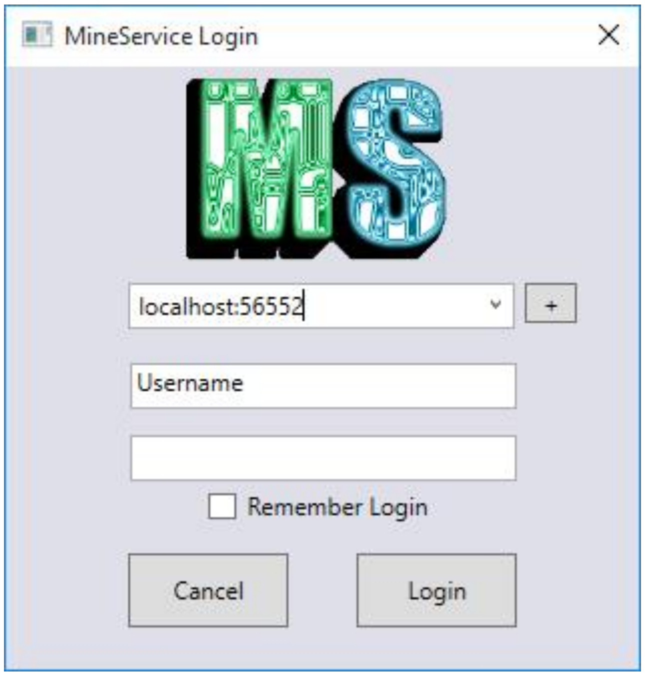
- This contains a sample MineCraft server used for testing in the Testing project.

User Guide

Adding a New User (In Progress):

To add a new user, simply add the desired username and salted password to the Users.db file.

User Login:



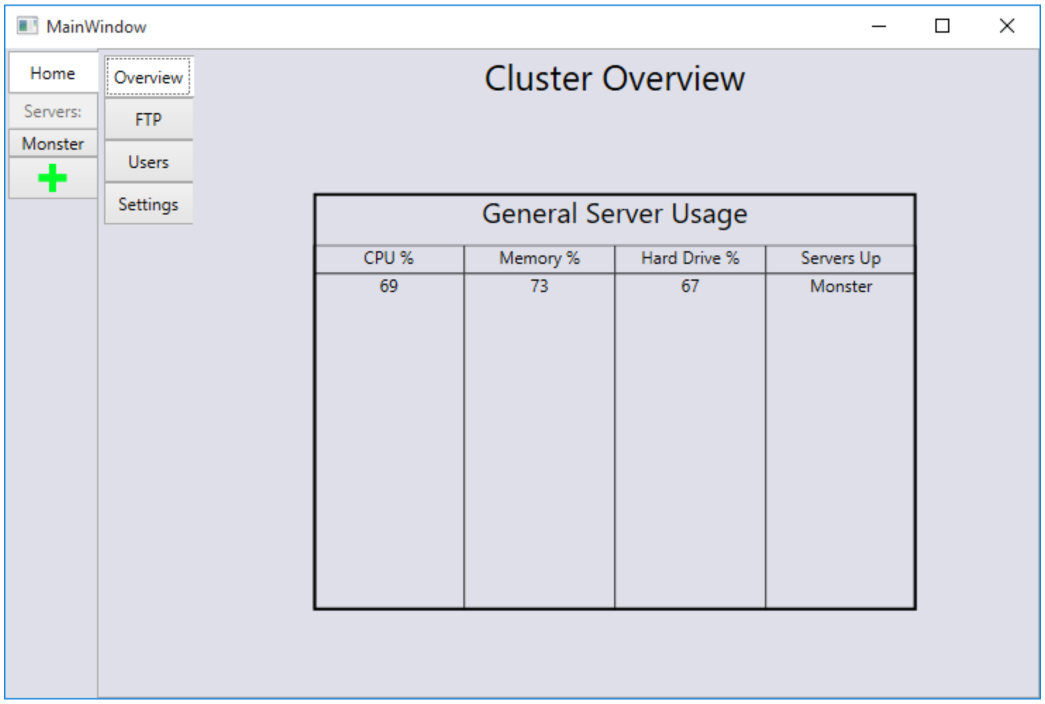
In the first text field put the server in the form of IP:Port (127.0.0.1:55555) that you wish to connect to.

In the next two fields put in the username and password given to you by your administrator.

Viewing the Home Tab:

The home tab contains general information about the cluster as long as anything that deals with elements on a cluster level such as settings and users.

Overview (In Progress)



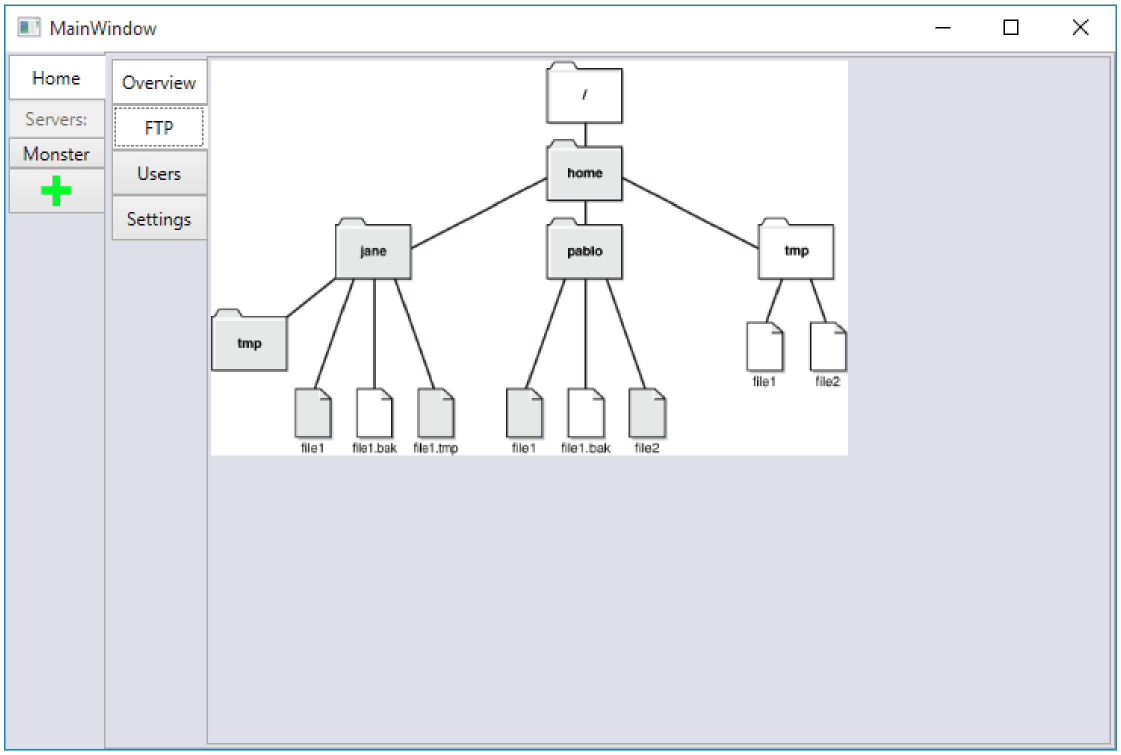
Cluster Overview

General Server Usage

CPU %	Memory %	Hard Drive %	Servers Up
69	73	67	Monster

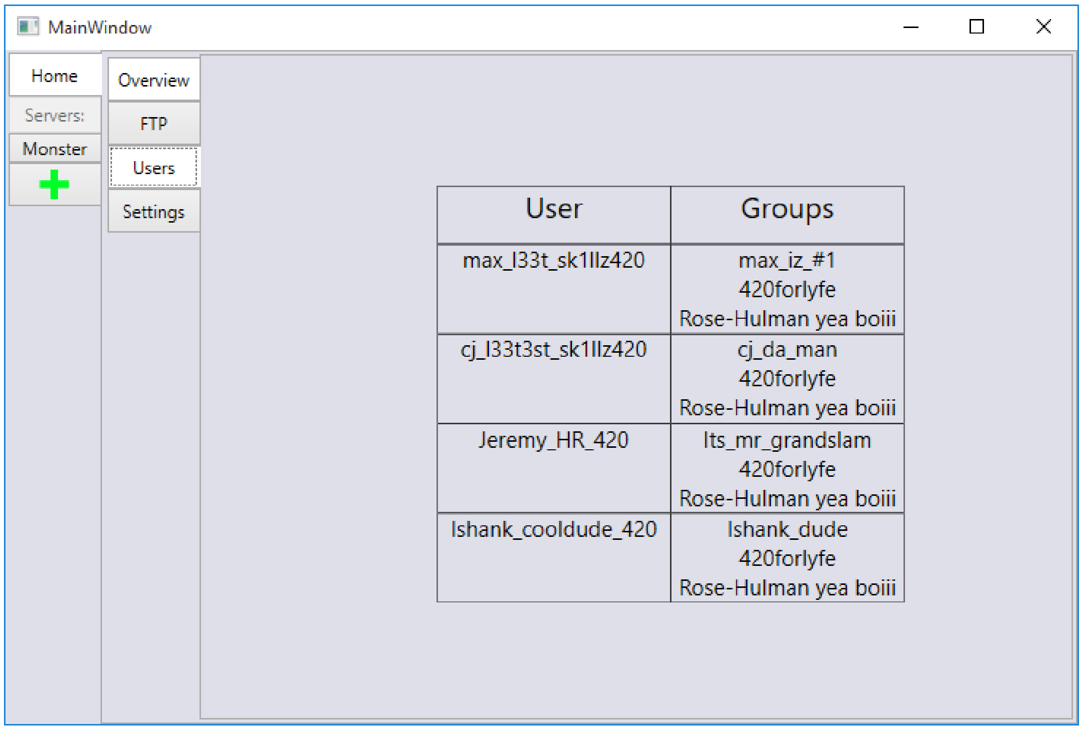
The overview tab is a quick way to see the general status of all the servers on this cluster

Viewing the FTP Tab (In Progress):



This tab will allow you to make changes to anything in the cluster file tree. This is for experienced users only and is restricted to clients who has full rights on cluster.

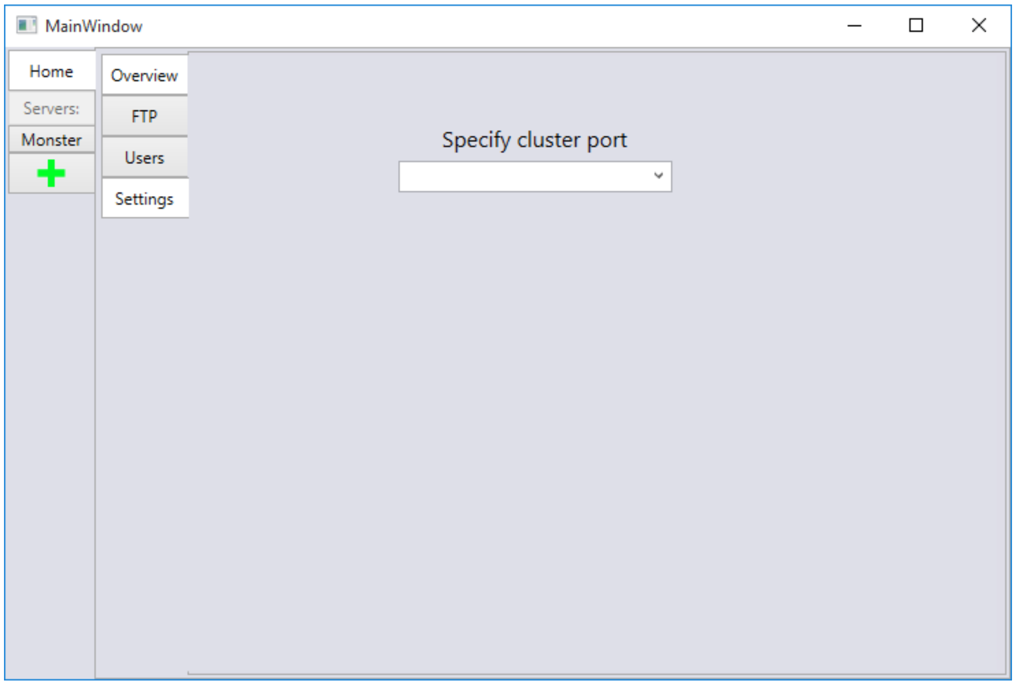
Viewing the Users Tab (In Progress):



User	Groups
max_l33t_sk1llz420	max_iz_#1 420forlyfe Rose-Hulman yea boiii
cj_l33t3st_sk1llz420	cj_da_man 420forlyfe Rose-Hulman yea boiii
Jeremy_HR_420	Its_mr_grandslam 420forlyfe Rose-Hulman yea boiii
Ishank_cooldude_420	Ishank_dude 420forlyfe Rose-Hulman yea boiii

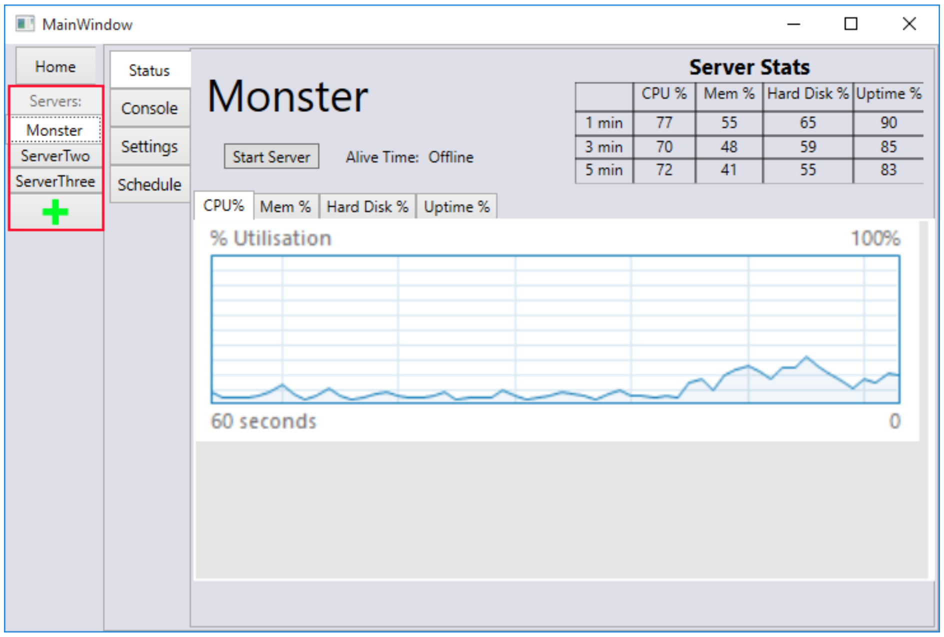
This tab will allow you to add and remove users as well as change their permissions.

Viewing the Settings Tab:



This tab contains the settings for the cluster. Currently only setting is the port number.

Viewing the Server Tabs:

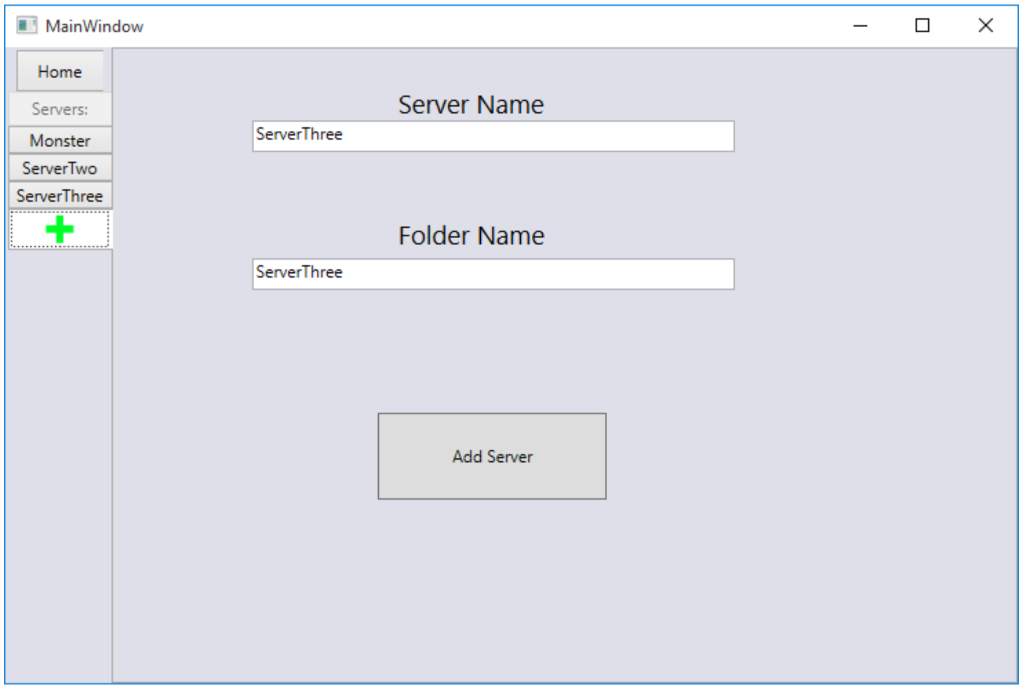


Server Stats

	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

The area marked in red is where any servers on this cluster will show up. The green + is where new servers are added.

Adding a New Server Tab:

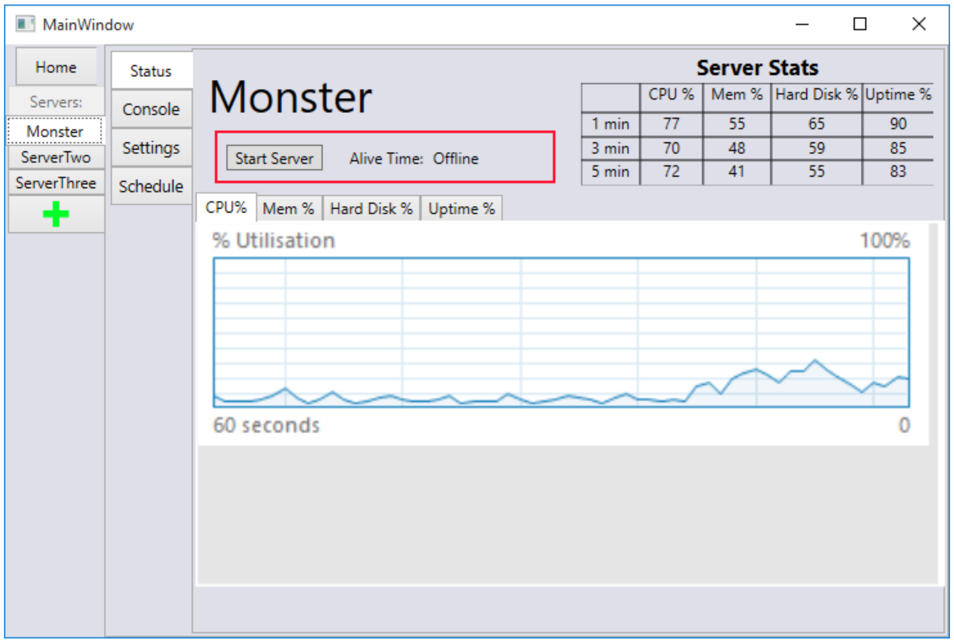


Server Name: ServerThree

Folder Name: ServerThree

Both the server name and folder name must be unique with what is on the cluster. After putting in two names and hitting add server it will show up with the other servers automatically.

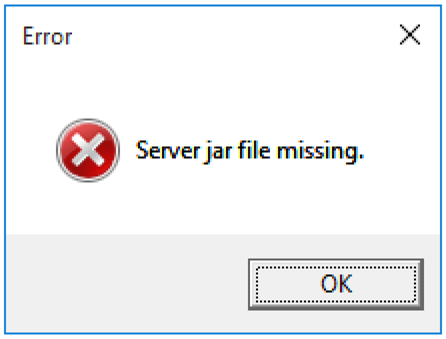
Viewing a Server:



Server Stats

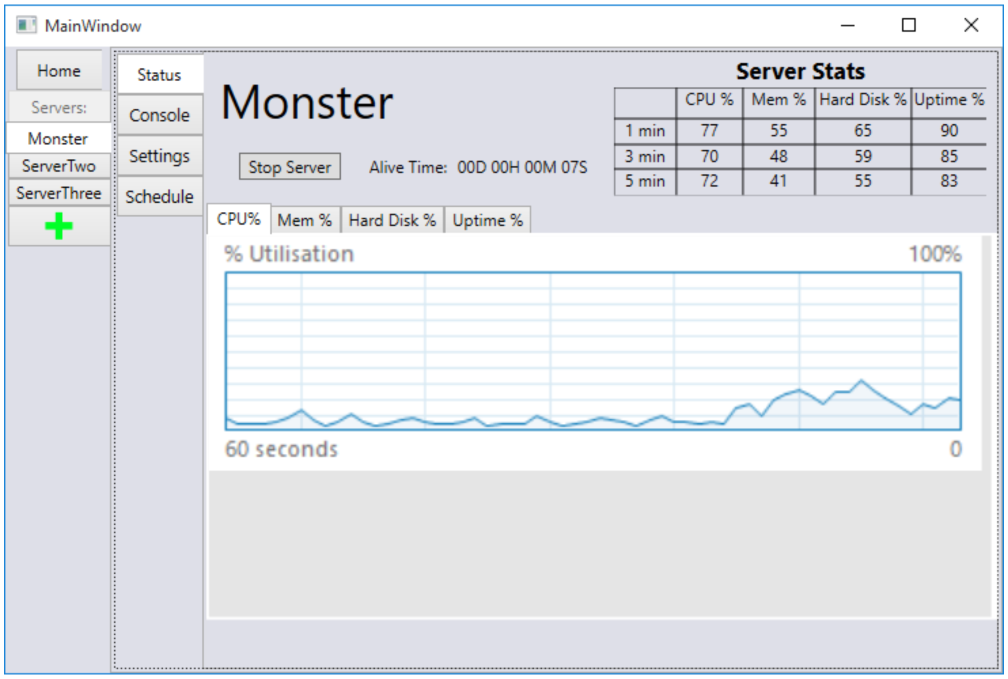
	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

Clicking start server will start the underlying Minecraft server.


Error

Server jar file missing.

Error will appear if there is no jar that matches the name set in the server settings.

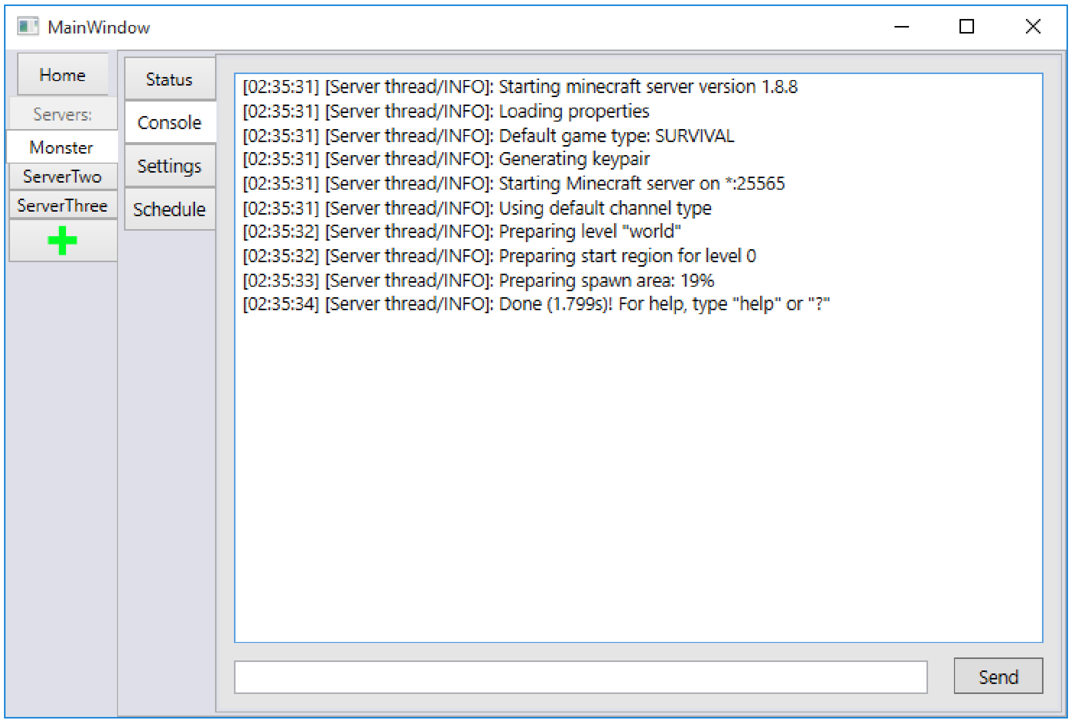


Server Stats

	CPU %	Mem %	Hard Disk %	Uptime %
1 min	77	55	65	90
3 min	70	48	59	85
5 min	72	41	55	83

After hitting “Star Server” it will tell the cluster to start the server. If it does start it will change to stop server and start showing the uptime.

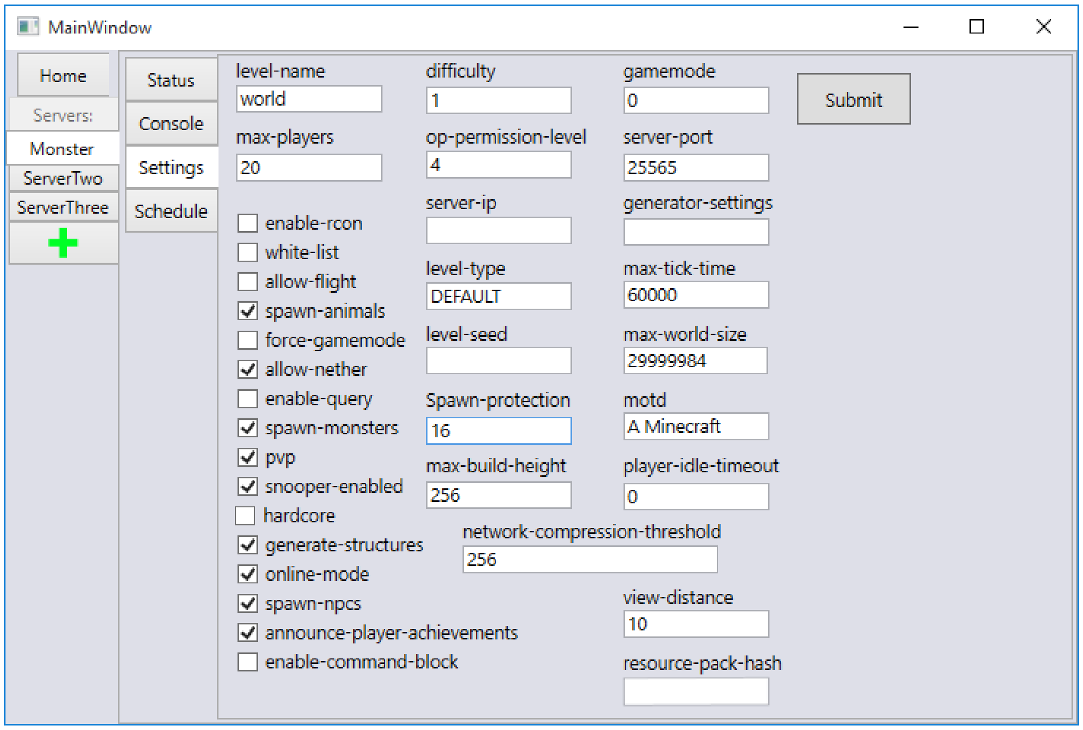
Viewing the Console:



```
[02:35:31] [Server thread/INFO]: Starting minecraft server version 1.8.8
[02:35:31] [Server thread/INFO]: Loading properties
[02:35:31] [Server thread/INFO]: Default game type: SURVIVAL
[02:35:31] [Server thread/INFO]: Generating keypair
[02:35:31] [Server thread/INFO]: Starting Minecraft server on *:25565
[02:35:31] [Server thread/INFO]: Using default channel type
[02:35:32] [Server thread/INFO]: Preparing level "world"
[02:35:32] [Server thread/INFO]: Preparing start region for level 0
[02:35:33] [Server thread/INFO]: Preparing spawn area: 19%
[02:35:34] [Server thread/INFO]: Done (1.799s)! For help, type "help" or "?"
```

This screen is a direct pipeline to the normal Minecraft Console. Anything inputted in the bottom will be feed directly to the Minecraft server upon hitting Send.

Selecting Server Settings (In Progress):



level-name: world
difficulty: 1
gamemode: 0
max-players: 20
op-permission-level: 4
server-port: 25565
server-ip:
generator-settings:
level-type: DEFAULT
max-tick-time: 60000
level-seed:
max-world-size: 29999984
Spawn-protection: 16
motd: A Minecraft
max-build-height: 256
player-idle-timeout: 0
network-compression-threshold: 256
view-distance: 10
resource-pack-hash:

- ☐ enable-rcon
- ☐ white-list
- ☐ allow-flight
- ☑ spawn-animals
- ☐ force-gamemode
- ☑ allow-nether
- ☐ enable-query
- ☑ spawn-monsters
- ☑ pvp
- ☑ snooper-enabled
- ☐ hardcore
- ☑ generate-structures
- ☑ online-mode
- ☑ spawn-npcs
- ☑ announce-player-achievements
- ☐ enable-command-block

On this page is all the normal settings found in the server.properties file in the Minecraft root directory.

Schedule (Not yet Developed)

Will allow for automatic restarts based on a time schedule.

Maintenance Guide

Installation

- Cluster:
  - Run setup.exe can be obtained from the MineService Website
  - The folder where the servers will be stored will be the same location as the program is ran from.
- Client
  - Run setup.exe can be obtained from the MineService Website or from your administrator.
  - Obtain the Server IP:port and username: password from your system administrator.
    - Starting entry username: “admin”: password: “password”

Updating

- Cluster
  - Will automatically check on startup and will ask you if you wish you update.
  - If the cluster is updated all clients should check to see if they need to update also.
- Client
  - Will automatically check on startup and will ask you if you wish you update.

    - Check to see if your current Cluster version will be compatible with the client update.

Troubleshooting

- Cluster Won’t start
    - Check the output from the console window. It should have details on what the issue is.
    - If that does not work delete the config.mineS
        - This will make all servers on the cluster seem to disappear. Just re-add the folder names from the client and all the files will populate the rest.
- Client wont connect
    - Check that you have an internet connection or at least connection to where the cluster is.
    - Confirm that the Server cluster is on.
    - If it is on check to make sure the port it is on is not blocked on the computer or anywhere in-between the two computers.
- Minecraft Server will not start
    - First check the console output via the client.
        - If it is printing out an error fix it.
    - Make sure eula.txt has the value eula=true in it
    - Make sure the Minecraft jar name watches what the MSMC.json file has in it.
    - Re-download the Minecraft Jar
        - Check the Hash of the downloaded jar