



Assignment Cover Letter

(Individual Work)

Student Information :

Given Name	: Jason Christian	Student ID Number	: 2301891714
Surname	: Hailianto		
Course Code	: COMP6510	Course Name	: Programming Languages
Class	: L2AC	Name of Lecturer	: Jude Joseph Lamug Martinez
Major	: Computer Science		
Title of Assignment	: GeminiFile		
Type of Assignment	: Final Project		
Due Date	: 20 - 6 - 2020	Submission Date	: 20 - 6 - 2020

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student :

A handwritten signature in black ink, appearing to be "JCH", written over a horizontal line.

Jason Christian Hailianto

1. Table of Contents

Plagiarism/Cheating	1
Declaration of Originality	1
1. Table of Contents	2
2. Introduction	3
3. Project Specification	4
4. Solution Design	5
3.1 Overview	5
3.2 Class Diagram	6
3.3 Everything Starts with a Command Line	6
3.4 Processing the arguments	8
3.5 Logger	8
3.6 Main Service	10
3.7 Peer Scanner	11
3.8 Local Communication Arguments Service	12
3.9 Message Wrapper Class	13
3.10 Peer Communication Managers	14
3.10.1 PeerServerManager	14
3.10.2 PeerClientManager	15
3.10.3 Double Connections	15
3.10.4 PeerCommunicationLoop	15
3.10.5 Message Processors	15
3.11 Binders	16
3.12 Binder Manager	16
3.13 On File Comparison	17
3.14 Net File Services	17
3.15 GUI Design	18
5. Working Program Evidence	22
6. Closing Remarks	25
7. Resources and References	26

2. Introduction

In the second semester at Binus International, we learned three more programming languages compared to the first semester. Knowing the basics of programming taught in the first semester, I have a personal philosophy to double down on what I have learned in the class, and to learn a lot more about programming and computer science concepts so that I can have more experience than yesterday. Because all of that, I have decided that for programming languages' final project, I would like to make the biggest project with the biggest variety in API I have ever undertaken. Now, I just need a good idea to make the application.

The idea came to me in early April. The idea of a file syncing application. I think that services like Google Drive, OneDrive, and other cloud-based services are only as fast as your internet connection and requires it. I also noticed that most people still use cable or a flash drive when they need to transfer files from a device to another. Sure, the speed is good but, you still need to connect things physically and you need to not happen to lose that flash drive you stored weeks ago in an obscure location. Then it came to my head that network file transfer is just as fast, and you don't need any physical connection to a device. It takes advantage of the router to transfer the file and nowadays, the cheapest routers have a speed up to 150mbps* which equates to 18.75MBps which is still fast and theoretically can download a 1GB file in under a minute. I took the concept even further and thought that it would be great if some devices can only sync a certain folder based on the user configurations.

To preface, I would like to say that this project GeminiFile is the biggest project that I have undertaken up until this point. I learned a lot of concepts not only in programming but in networking, multithreading, java project structures, data streams, schedulers, executors, and a lot more. There will be a lot of imperfections in the program and concept implementations as I am still learning all of these. Just know that I did my best and I did all I could in these past two months developing the application.

The project officially began on the 23rd of April and the first binaries are released in GitHub in 17th of June. This project is coded in Java and built using the amazing IntelliJ IDE by JetBrains. This project is open source in nature, and is available at my GitHub repository <https://github.com/nakamarusun/GeminiFile/>.

This program has been successfully tested with 3 concurrent connections from 2 Windows based computer, and a Raspberry Pi Zero server running on Raspbian (Debian GNU/Linux).

The demonstration video is uploaded to youtube, and can be viewed from this link: <https://youtu.be/s1vzhnzRAbM>

3. Project Specification

Project purpose:

To make multi-device file syncing easier, not require physical connections and accessory devices. Furthermore, create a folder that can only be synchronized to certain devices with the same folder id.

Project audience:

Everyone, anyone that needs to have one or more folders to be synchronized with one or multiple devices to create an integrated home file ecosystem.

Project aim:

Create a functional program that can syncs files from a designated folder automatically to another device that has the same folder id. The program should run in the background without hassle, and automatically detects changes in the folder to compare it with another device.

Project Requirements:

- Peer-to-Peer connection with other devices that run the GeminiFile service.
- TCP socket communications to receive and send information to decide what peers need.
- Auto peer discovery using a ping-based scanner system.
- Works on PC, Mac, Linux
- Syncs folder with another peer based on a mutual folder ID.
- Folder watcher to detect any changes
- Block-divided file transfer system.
- Multithreaded file transfer.
- Program function logs to a file and console.

4. Solution Design

3.1 Overview



Image 3.1.1. GeminiFile logo.

GeminiFile uses two external libraries¹, the JSON-Java library to convert classes into JSON files for safekeeping and message protocols, and JavaFX to create the final GUI. The rest is purely from the excellent extensive native java library. These include, but not limited to:

- Threads and Runnable
- Loggers
- Sockets and Server Sockets
- Input / Output Stream
- Regular Expression
- Executor Services
- Scheduled Executors
- Locks and Reentrant Locks
- Watch Services
- Message Digest

3.2 Class Diagram

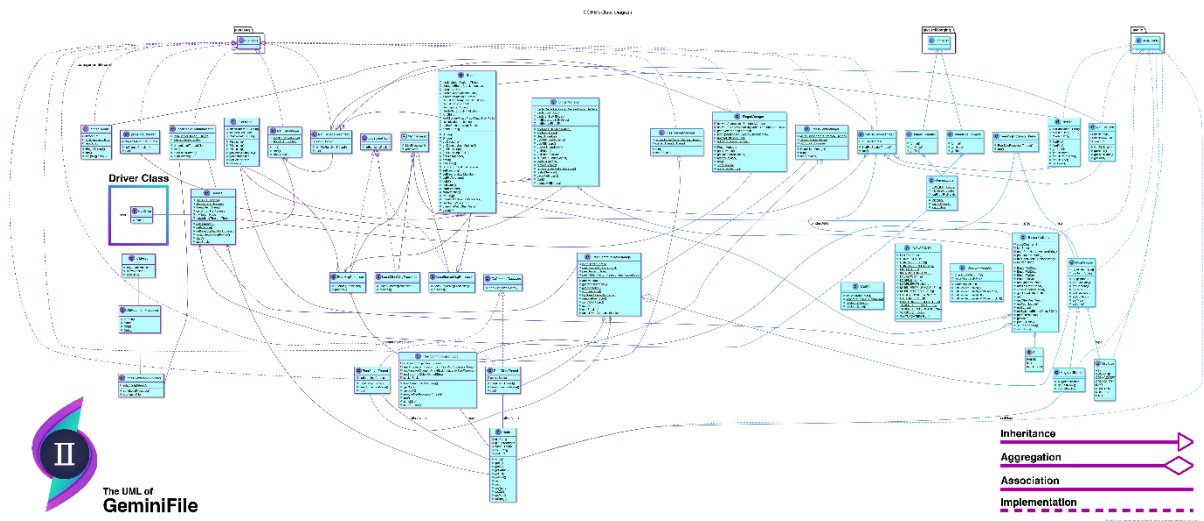


Image 3.2.1. GeminiFile CLI Class Diagram

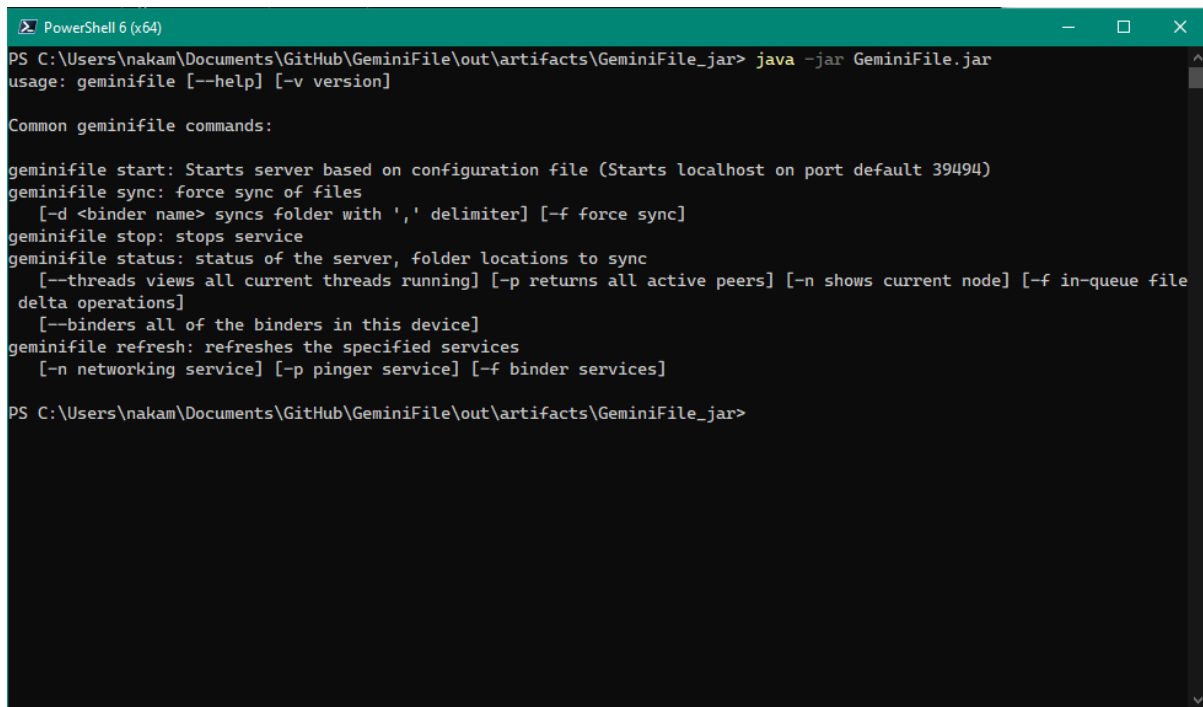
Pictured is the UML Diagram for the CLI version of GeminiFile. The image file is huge and it's not possible to display all the classes and connections in word. The file can be viewed in its' raw form by saving the above picture to the disk, or by downloading it from here: <https://raw.githubusercontent.com/nakamarusun/geminifile/master/artfiles/UMLNew.png>.

3.3 Everything Starts with a Command Line

From here on out, the code explanation begins. Before developing a GUI version, naturally I want to make a fully working program first that runs in the terminal. Almost all CLI applications runs with arguments that you can put next to the main executable name. So first, I must make a working system where the program can process the words and commands, discard bad arguments, and detects errors when user types in the terminal.

When I think about how to process the arguments, I suddenly remembered my first semester Discrete Structures course with Sir Fergy, where he explained something called regular expressions. I remembered that he said something along the lines of that it can be used to process “words, phrases, and languages”. Therefore, I decided to check it out. It turns out that RegEx is perfect to process arguments for my program, so I used it.

For GeminiFile, I used git as a main point of observation and inspiration for my implementation of the solution. Notice if you only type “git” in the terminal, the interface does not do any commands instead, it shows the help page, where you can see and know what commands does what the next time you use git. I thought this is a good idea so that the user will not accidentally starts or does anything with the GeminiFile service, so I implemented the same core idea.



```
PowerShell 6 (x64)
PS C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar> java -jar GeminiFile.jar
usage: geminifile [--help] [-v version]

Common geminifile commands:

geminifile start: Starts server based on configuration file (Starts localhost on port default 39494)
geminifile sync: force sync of files
    [-d <binder name> syncs folder with ',' delimiter] [-f force sync]
geminifile stop: stops service
geminifile status: status of the server, folder locations to sync
    [--threads views all current threads running] [-p returns all active peers] [-n shows current node] [-f in-queue file
    delta operations]
    [--binders all of the binders in this device]
geminifile refresh: refreshes the specified services
    [-n networking service] [-p pinger service] [-f binder services]

PS C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar>
```

Image 3.3.1. GeminiFile commands

This is what will happen if you type “java -jar GeminiFile” or “java -jar GeminiFile --help” to the command line. A miniature help page appears to help you understand a little bit of the service. This is a comprehensive list of what commands does what in detail:

- **Start:** Starts the GeminiFile service, simple enough. This service will be explained in detail later in the report. But to make it short, all the GeminiFile services will be started and the program will automatically do its job in the background without any intervention. When the terminal is closed, the service will stop too.
- **Sync:** In itself, this won’t do anything without extra arguments.
 - -d <binder name>: Syncs the folder name that you specify with another peers that shares the same folder id.
 - -f: forces the service to sync all the folders with every available peer with the same folder id.
- **Stop:** Stops the service.
- **Status:** Shows the status of the service.
 - --threads: shows all the running threads in the service, with name.
 - -p: displays all the peers this service is connected to.
 - -n: shows the detail of this device in GeminiFile network terms.
 - -f: shows all the current upload / download operations in this device.
- **Refresh:** In itself, this won’t do anything without extra arguments.
 - -n: Restarts all the networking service. All current operations will be stopped.
 - -p: Scans the entire network for new peers immediately.
 - -f: Restarts the service which controls the folder watchers.

3.4 Processing the arguments

When the GeminiFile service starts, one would notice that the terminal is blocked, and they cannot type any additional commands. So, one would think, what could they do to let's say view the status of the running service. When the GeminiFile service starts, it also starts up a localhost server, where if users launch up another terminal, they can type the other arguments to view the status of the running GeminiFile service.

When "GeminiFile status" command is run, the current terminal will attempt to connect by TCP to the currently running GeminiFile service. After connecting, the service will then send a message request of what the service should reply and display to the end user. After the main service has replied and the user has received it, the connection will close, and the user will have viewed the status message sent from the service.

This works by passing the arguments from the main CLIDriver class to CLIArgumentProcessor, where it will be turned into the message that will be sent to the main service using the LocalClientCommunicator class. The same LocalClientCommunicator will receive a reply from the main service and will process it before displaying all the information back to the user.

3.5 Logger

It is important to display necessary info to the user about all the current operation the GeminiFile service is doing. Java's System.out.println() is not sufficient to this program's need because, of the lack of message customizability. The GeminiFile program should display the time, and the level of importance every message outputted by the terminal to make the user understand more clearly. That's why java Logger is used to display all the messages. Loggers can be customized to have custom output message complete with dates and every exception encountered to easily debug and understand the flow of application while in development. It also allows output to a file to make life easier.

Both the file and console logger can be written to simultaneously because of the nature of loggers that allows multiple custom handlers to be added. Furthermore, those handlers can be customized with a formatter.


```
PowerShell 6 (x64)

#####
###
GeminiFile v0.0.1 by Jason Christian

PS C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar> java -jar GeminiFile.jar start
{GeminiFile log v0.0.1}
{INFO} - Service is starting...
{INFO} - Networking service is starting...
{INFO} - [LSERVER] Opening in localhost:43743
{INFO} - ip is: 192.168.1.12
{INFO} - [PING] Pinger service is starting...
{INFO} - [PEER] Starting Peer Communicator Manager...
{INFO} - [FILE] Binder Manager is Starting...
{INFO} - Loaded binders JSON from:C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar\MyBinders.json
{INFO} - [Binder] Starting directory watcher TestBinder
{INFO} - [NetFile] Starting file receiver service...
{INFO} - Registered C:\Users\nakam\Desktop\TestSync to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\New folder to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\Please do something about this to directory watcher
{INFO} - 192.168.1.12:54638 Is open!
{INFO} - [PING] Process done after: 8849
{INFO} - [PING] Process will restart after: 21144
{INFO} - 192.168.1.12:54638 Is open!
{INFO} - [PING] Process done after: 8819
{INFO} - [PING] Process will restart after: 21180
PS C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar>
```

Image 3.5.1. Logger outputting to terminal in the CLI service.

```
C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar\gemini.log - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

RandomUtil.h MyBinders.json MyBinders.json GeminiGUI.css new 1 settings.json gemini.log new 2 bindercell.fxml LaunchGeminiFile.sh

1 [GeminiFile log v0.0.1 at Thu Jun 18 22:09:59 ICT 2020 ]
2
3 [18/Jun/2020 22:09:59] INFO - Service is starting...
4 [18/Jun/2020 22:09:59] INFO - Networking service is starting...
5 [18/Jun/2020 22:09:59] INFO - [LSERVER] Opening in localhost:43743
6 [18/Jun/2020 22:10:00] INFO - ip is: 192.168.1.12
7 [18/Jun/2020 22:10:00] INFO - [PING] Pinger service is starting...
8 [18/Jun/2020 22:10:00] INFO - [PEER] Starting Peer Communicator Manager...
9 [18/Jun/2020 22:10:00] INFO - [FILE] Binder Manager is Starting...
10 [18/Jun/2020 22:10:00] INFO - Loaded binders JSON from:C:\Users\nakam\Documents\GitHub\GeminiFile\out\artifac
11 [18/Jun/2020 22:10:00] INFO - [Binder] Starting directory watcher TestBinder
12 [18/Jun/2020 22:10:00] INFO - [NetFile] Starting file receiver service...
13 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync to directory watcher
14 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync\New folder to directory watcher
15 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan to directory watc
16 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\New folder to dir
17 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag to directo
18 [18/Jun/2020 22:10:00] INFO - Registered C:\Users\nakam\Desktop\TestSync\New folder\Please do something about
19 [18/Jun/2020 22:10:00] INFO - 192.168.1.12:54638 Is open!
20 [18/Jun/2020 22:10:09] INFO - [PING] Process done after: 8849
21 [18/Jun/2020 22:10:09] INFO - [PING] Process will restart after: 21144
22 [18/Jun/2020 22:10:30] INFO - 192.168.1.12:54638 Is open!
23 [18/Jun/2020 22:10:39] INFO - [PING] Process done after: 8819
24 [18/Jun/2020 22:10:39] INFO - [PING] Process will restart after: 21180
25
26 [End of GeminiFile log v0.0.1 at Thu Jun 18 22:10:40 ICT 2020 ]
27

Normal text file length: 1,935 lines: 27 Ln: 16 Col: 1 Sel: 253 | 4 Unix (LF) UTF-8 INS
```

Image 3.5.2. Logger outputting to a log file.

3.6 Main Service

Now is the main lifeline of GeminiFile; all the services that it runs to make the file syncing possible.

When the service first starts, the program will create a self-identity based on the machine's OS, Hostname, IP, port, unique identifier, so that it can communicate with other machine with the current identity. Before starting all the sub-services, this service also checks whether the device is connected to a network. If not, then the service will wait until a network connection is detected.

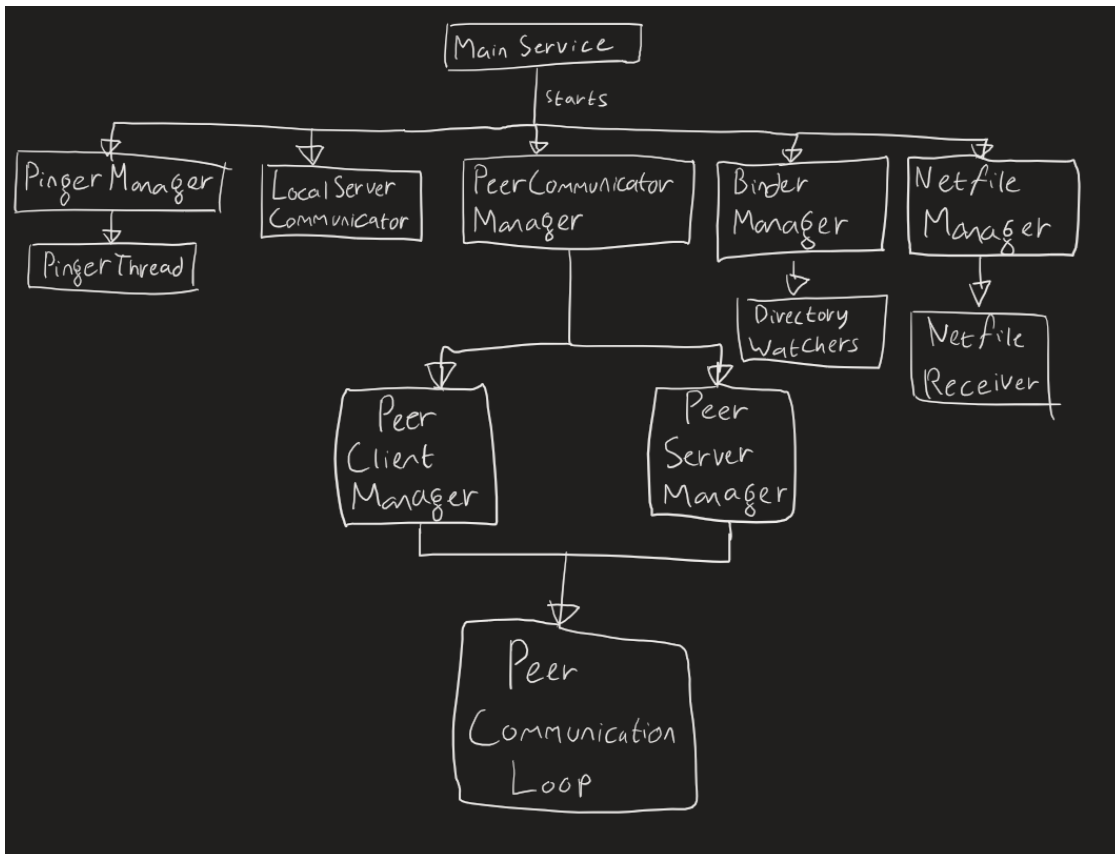


Image 3.6.1. Main Service Diagram

In total, the main service's job is to start 5 other integral services needed to make the program function properly, and to handle interrupts thrown to restart or stop one or more of the services. The individual services will be explained in great detail later in the report. Here is a quick overview of what each service does:

- **PingerManager:** Scans the network for new peers to connect to in each interval. Does this by pinging all the IP addresses one by one to find an open GeminiFile port and confirms the port is open because of GeminiFile, and not others.
- **LocalServerCommunicator:** Opens a local port for CLI command inputted by the user. Processes the message and possibly returns a reply.
- **PeerCommunicatorManager:** Handles a socket server for connection and client sockets to connect with others. Both services act as a peer socket, establishing

connection with other devices on the network. Sends and receives communication and requests based on the needs of the current machine. Based on the communications this service will process the messages and turns it into an instruction.

- **BinderManager:** Starts up every directory watcher in folders controlled by GeminiFile, to detect changes and communicates to other peers based on the changes.
- **NetFileManager:** For every file sync operation received and confirmed by the device, this service serves to receive incoming files in form of blocks and converts them into a usable file in the temporary folder. Then, moves it into the main folder.

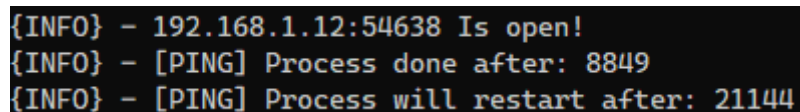
3.7 Peer Scanner

When multiple devices start up a GeminiFile service, they need a way to find each other in a network, and to confirm it is indeed another GeminiFile service. One way the program can do such a thing is by manually inputting the other device's IP address and connecting to it. Sure, that's a good way and needed because it is a manual override. However, to be truly hassle-free and to run independent of any intervention, the program needs a way to find each other automatically in the network.

An automatic scanner that runs periodically solves this problem. The PingerManager class manages to scan all the available IP in a network, from x.x.x.1 until x.x.x.255. Obviously, to scan 255s IP is a lot. And to scan one, the program needs to wait a certain value before deciding that the IP is in fact not open. This is called timeout value, and it can be in order of hundreds of milliseconds based on the configuration. To speed up the scanning processes, parallelization is important to divide up the tasks, and in result, faster results.

After scanning and getting all the new peers that the current device is currently not connected with, the PingerManager class will send a request to PeerClientManager to try to query a connection to the aforementioned peer.

This service runs on an interval in the configurations, and managed by java's ScheduledExecutorServices.



```
{INFO} - 192.168.1.12:54638 Is open!
{INFO} - [PING] Process done after: 8849
{INFO} - [PING] Process will restart after: 21144
```

Image 3.7.1. Peer scanner outputs

3.8 Local Communication Arguments Service

```
{INFO} - [LSERVER] Connected !  
{INFO} - [LCLIENT] {ASK: threads with 0 bytes in data.}  
{SEVERE} - [LSERVER] Client Disconnected.
```

Image 3.8.1. Main service receives a request from CLI argument.

This service functions as a message receiver from the inputted CLI argument from another command line that the user inputs. The service processes the message received, and asks the necessary things needed by the user from other services, then constructs them into a string for the user to see. After the string is fully constructed, a new message will be constructed and sent back to the CLI program.

To receive these messages, the service needs to open a communication port in the loopback IP address. This loopback IP address is not open for the whole network to see but is isolated within the machine. Because the nature of this isolated connection, it does not need to be secured. The message sent back and forth through this connection is an object created by the MsgWrapper class. This class contains the message and the type of message it is. More will be explained.

```
[LCLIENT] Connecting to localhost:43743  
[LSERVER] {INFO:  
Current running threads on GeminiFile Service:  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Signal Dispatcher  RUNNABLE    9 Daemon  
NetworkChangeDetector  TIMED_WAITING 5 Normal  
Pinger          WAITING      5 Daemon  
FileSystemWatchService  RUNNABLE    5 Daemon  
Pinger          WAITING      5 Daemon  
BinderWatcher-ASjd5    WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
NetFileReceiver  RUNNABLE    5 Normal  
Pinger          WAITING      5 Daemon  
LocalhostMessageServer  RUNNABLE    5 Normal  
PingerManager    WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          TIMED_WAITING 5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
PeerClientManager  RUNNABLE    5 Normal  
Pinger          WAITING      5 Daemon  
Attach Listener  RUNNABLE    5 Daemon  
Common-Cleaner    TIMED_WAITING 8 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Reference Handler  RUNNABLE    10 Daemon  
PeerServerManager  WAITING      5 Normal  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
Finalizer         WAITING      8 Daemon  
Pinger          WAITING      5 Daemon  
main             WAITING      5 Normal  
Pinger          WAITING      5 Daemon  
Pinger          WAITING      5 Daemon  
with 0 bytes in data.}
```

Image 3.8.2. Example of a reply received from the main GeminiFile service.

3.9 Message Wrapper Class

Every communication handled by this program either for local argument processing, or peer communications needs a fixed messaging class.

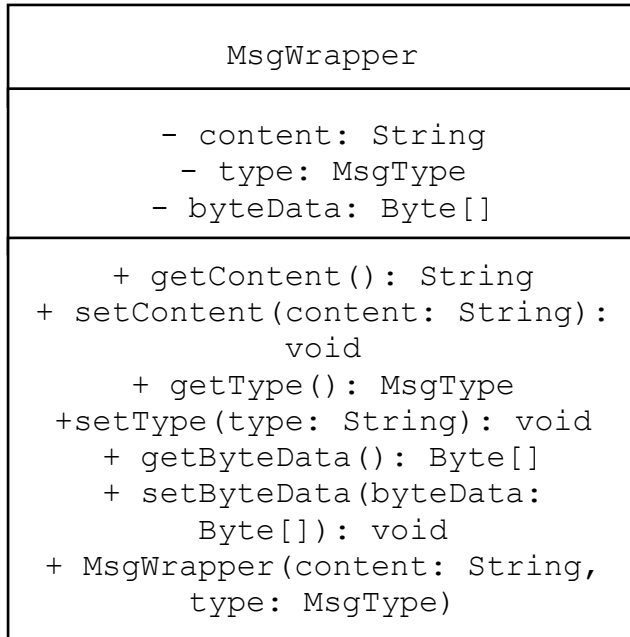


Image 3.9.1. MsgWrapper UML Diagram.

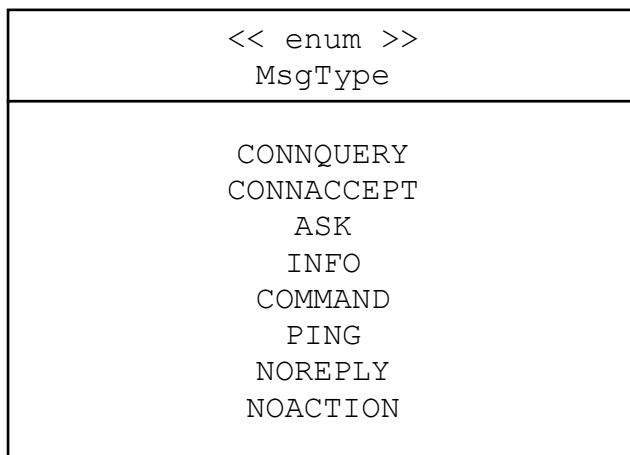


Image 3.9.2. MsgType enumeration for message wrapper

Every communication handled by this program either for local argument processing, or peer communications needs a messaging class to wrap the message. Message wrappers are beneficial so that the program can sort and easily understand what each message contents are for.

The message wrapper class contains 3 fields. The content field is the main message where the program can process it and does a command depending on what the message is. The message type field dictates the type of message it is. The explanation are as follows:

- **CONNQUERY:** Asks the other peer this message is directed towards for a new connection with the current device.
- **CONNACCEPT:** Means that the message has been verified, and the connection is approved by the device.
- **ASK:** Sends the device a generic ask command. Will expect a reply after sending this message.
- **INFO:** Sends the device an informational string. The string will then be automatically printed to all the assigned logger.
- **COMMAND:** Sends the peer a command message. The peer will do the command and replies a status whether the command has been successfully done.
- **PING:** When this message is sent to the peer, the peer will reply a similar message.
- **NOREPLY:** A message type that when sent, does not expect a reply immediately.
- **NOACTION:** A special type of message that instructs the program not to send a message.

These message types and wrappers are an integral part of the application and makes sure that every communication sent back and forth are organized and dynamic.

3.10 Peer Communication Managers

In the program, it is important for peers to be able to communicate with each other to do essential operations, and to do targeted file syncing. These communications are of course not possible without a proper connection protocol with the other peers that this service is trying to communicate.

These connections are made possible with all the available peer data gotten from the previous peer scanner service. The `PeerCommunicationManager` can be divided into two subclasses that manages these connections but diverges into one `PeerCommunicationLoop` object in the end to simulate true peer-to-peer connection, and not a client-server connection.

Before connecting, the services also manage and verifies the other device's identity based on the identity's creation in the beginning of the main service. After successfully connecting to the peer, the identity is then added to the main service's peer table, to keep reference of all the connections it has made.

3.10.1 PeerServerManager

This sub-service purpose is to send a new connection queries to other devices from all the available and unconnected IPs gotten from the scanner service. This service will be idle and wait for a new IP to be available from the Scanner through the `ArrayBlockingQueue` class from java, where if the queue is empty, the program will block itself until a new value is available. After getting the new IP, the service will try

to attempt a connection with the other peer through the CONNQUERY message type. This type of query will be accepted by PeerClientManager

If the new connection is accepted by the other machine, then this newly established socket connection will be transferred to PeerCommunicationLoop, where the connection will simulate a peer-to-peer connection with the device. In addition to that, the new device will be added to the services peer table.

3.10.2 PeerClientManager

This sub-service will wait and receive incoming connections from other devices on the network sent by the aforementioned PeerServerManager. After the connection has been verified, the newly made socket connection will also be transferred to a new PeerCommunicationLoop. The new connection will also be inserted to this services peer table.

3.10.3 Double Connections

In program testing, it is found that two simultaneous connections queries from two different devices will result in a double peer connection. To prevent this type of bug, Both the peer client and server managers implements a thread lock. This lock is enforced so that only one connection queries or acceptance can happen at a time.

In cases where simultaneous connection still happens, the java lock implementation will choose to let one thread to run, while blocking the other thread until the running thread has completed its' operation or run into any exceptions.

3.10.4 PeerCommunicationLoop

Every connection accepted and made will be made into this class, where all the back and forth high-level communication will occur. This class has its own in and out socket streams to make this possible. For every peer with accepted connections, there should be only one of this objects that handles that communications.

To receive and send messages simultaneously, this object will run one extra thread so that I/O can happen without any interruptions.

3.10.5 Message Processors

Now that the message communication protocols are done, the message needs to be processed independent of the communication threads. This is so that the message processing bit will not interfere at all with the communications, to minimize any connection hiccups. This is handled by the PeerMsgProcessor class, and LocalServerMsgProcessor for the local argument processing.

Each class contains all the directives needed to be done when receiving a certain message with a certain content.

3.11 Binders

In this program, a folder that is managed and watched by GeminiFile is called a “binder”. But, for this report, the term folder will be used instead of binder. Binder is a class used in this service to keep a list of folders that the user wants to be synced with other devices. There are a lot of fields in this class and therefore, only the essential ones are going to be explained.

The field “name” is used only in the machine to provide the user a clearer identification aside from its’ ID, it functions no more than that. The field “directory” contains the path to the folder the binder manages. Every folder has a unique identifier in the “id” field. This is the most important field because it is used in knowing which folders are mutual in the synchronization process. Essentially, what happens in the syncing process when the service calls for it in the main service is, two or more peers communicates what IDs they have in common. If there are no common IDs, the syncing will stop. However, if a mutual folder ID is found, then both machines will carry on with the process to compare what files they have in common, and what they need. In simple terms, folders with the same id in different devices will be synced by the service to have the same files inside, regardless of path and name.

3.12 Binder Manager

This service manages all the folder operations. First, this service attempts to load a json configuration file that contains all the folders managed by the GeminiFile service. If it does not find the file, a new empty json file will be created by this service, and this service will start with no folders attached to it.

Provided that the configuration file is not empty, or a new binder is added to the program, the service will send a sync request to every other peer with the same binder ID with the intention to sync files. The message processor will then compare all the files based on their MD5 hash sum first, then checks the last modified times. After the comparison, the main service will decide which peers would need which files, and then the service will start sending and receiving all the needed file based on the operations.

This service will also start a directory watcher for all the registered folders. These watchers are a part of java’s API named WatchService. As the name suggests, these services watches over a folder for any modifications, new and deletion processes invoked by the user in said directory. For folders inside a folder, a new watch service instance must be registered

also. Therefore, all the watchers need to be created recursively so every folder in the GeminiFile watched folder has a watcher registered to it. These watchers will block the thread, until an action has been detected, and will continue the code block. The service will then take note of every modifications in the folder, namely the file name, and the path to a java collection variable, then waits for another interval for changes. After a certain interval, all the changes detected in the interval will then be staged for syncing to all the other peers.

3.13 On File Comparison

To know which files needed by the other peer and the current device, the program needs a concrete algorithm to decide that actions. To compare files first, all the files in the folder are loaded into the java File class. Then, the following metadata are extracted from the files:

- Last Modified: The last time the file is modified by the user.
- MD5 checksum: A special 32 bytes string that represents the integrity of the file.

First, when the file comparison algorithm detects a file which has the same name in the same directory, it will try to compare the MD5 checksum string. If both files have the same checksum, then both files do not need to be modified and synced by the service. However, if both checksums are the same, then the service will choose the file with the most recent date of modification.

These comparisons are then put into a new BinderFileDelta, where the object stores what the other peer and this peer needs in terms of file. After negotiation between the two peers are complete, and the file transfer is approved by both the peers, the delta operation will then be executed, and files will be transferred between the two devices concurrently.

3.14 Net File Services

In this service, incoming connections to receive files are handled. After a delta operation is handled by both devices and approved, then both the peers need to query a connection to the opposite net file services. To send a file over to the other side, a device needs to send the following file's metadata over. These metadata are handled by the class NetFile, and it includes the file name, path, and size. After sending the metadata, the actual file will be sent over in forms of blocks. When the file is big, it is not practical to send all of it in one go, because it will be a resource hog. In this program's implementation, any files are separated into multiple 4096 bytes blocks. The number of blocks needed to be sent over to the other device is dictated by this simple math:

$$Blocks = \left\lceil \frac{File\ size}{Block\ size} \right\rceil$$

To not interfere directly with the main file the service is currently downloading towards, the file will be first put into a temporary folder made by GeminiFile. After successfully downloading all the necessary blocks to build the application, the application will attempt to move the file into the main directory where the file is supposed to be. If the move operation failed, the file will remain in temporary directory.

After all the files are successfully transferred and received, the delta operations will be complete and be closed.

3.15 GUI Design

All the GUI programming begins in the 13th of June 2020. Prior to that, it has been all full CLI programming. Before said date, I have not been familiar with the JavaFX API.

For the GUI in GeminiFile, JavaFX is used. All the layouts are made using the Scene Builder by Gluon. The program produces .fxml file formats which then can be loaded by the JavaFX API using class streams. The GUI is then constructed and showed in the screen. A new driver class is made for loading all the GUI, but you can still use the standard CLI commands by invoking it in the terminal argument.

Before designing and coding the GUI program, I designed the logo for GeminiFile. For the design, I used purple and cyan to represent the primary color palette for the program. It's called GeminiFile because, it duplicates file and keeps the same duplicate in each device. Because the nature of that, I used two main colors too, like twins.

Then after making the logo, I have an idea to make an animation in the main app that can represent the current state of the GeminiFile service interactively. I looked at documentations online and found that you can do animations in JavaFX pretty well. For the animations, I used the RotateTransition class in JavaFX.

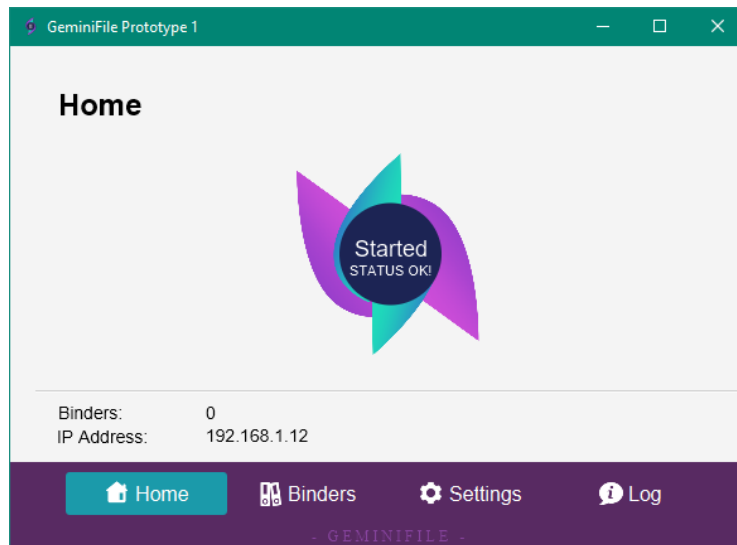


Image 3.15.1. GeminiFile Home (it spins!)

The main interface is a single page application, and has an empty space in the middle, above the navigation bar. We will call the middle part canvas for ease. The canvas can be later filled with another independently made .FXML files made once again in the Gluon Scene Builder. The canvas of course, will change depending on what the user clicks on the navigation bar. Home is the default option. All items in the navigation bar is a toggle button controlled by JavaFX's ToggleGroup. This allows the program to have only one button active at any time. More code is implemented to ensure that one button will always be active at any time.

In the home page, a button is located at the center of the canvas to start and stop the service. Both the flaps will start to spin in random direction and speed when the user clicked the start button to signify the service has been started and is ok.

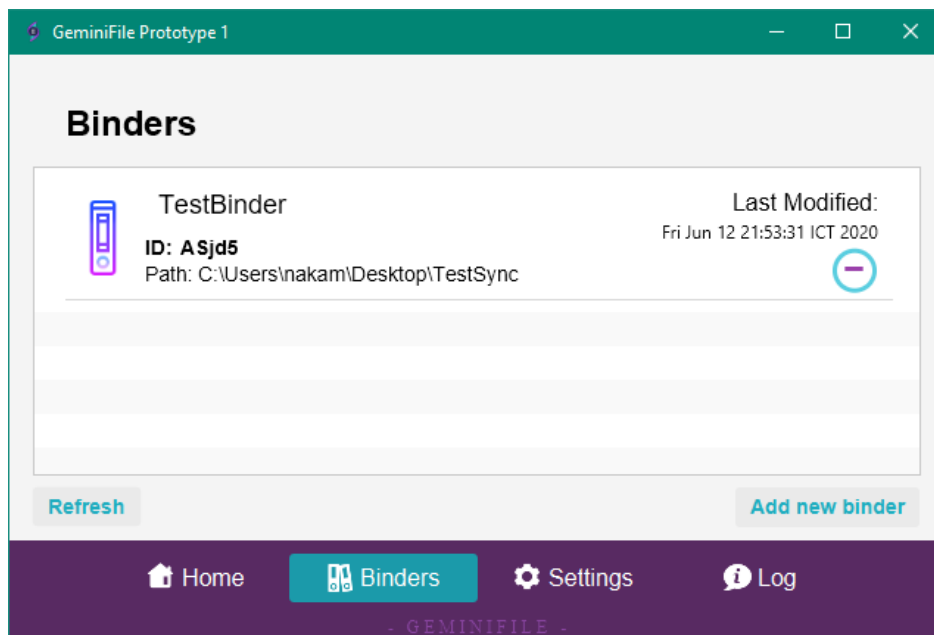
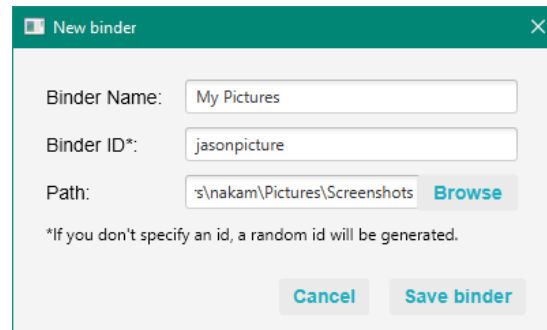


Image 3.15.2. Binders page

In the binders page, you can see all the folders managed and watched by the GeminiFile service. These folders will only appear only if you have started the service before, and all the

buttons in this canvas will be disabled if you have not. The add new binder button, as the name suggests, is used to add a new folder to be watched by the GeminiFile service. The minus button in the each of the binders can be clicked to unregister the folder from the main service. Clicking that will open a new window that can be used to enter the details of the binder that will be created.



The image shows a 'New binder' dialog box with a title bar containing a close button. Inside the dialog, there are three input fields: 'Binder Name' with the text 'My Pictures', 'Binder ID*' with the text 'jasonpicture', and 'Path' with the text 's:\nakam\Pictures\Screenshots'. To the right of the 'Path' field is a 'Browse' button. Below the input fields is a note: '*If you don't specify an id, a random id will be generated.' At the bottom of the dialog are two buttons: 'Cancel' and 'Save binder'.

Image 3.15.3. Prompt to create a new binder.

You can enter all the necessary details to create the new binder. Note that this form requires the binder name and path to be filled and will be check for its' validity. Otherwise, the program will throw an error and the user needs to refill the form with the correct data. The binder ID can be left empty, and the program will generate a random 7-byte string to be used. The browse button will open a new window so that the user can browse directories in the machine and select the right folder. The path can also be typed or pasted manually, but of course it will be checked by the program for its' availability.

To show all the binders, the ListView class is used to display a collection of binders in the screen. Custom content in the ListView is made inside the Gluon Scene Builder too.

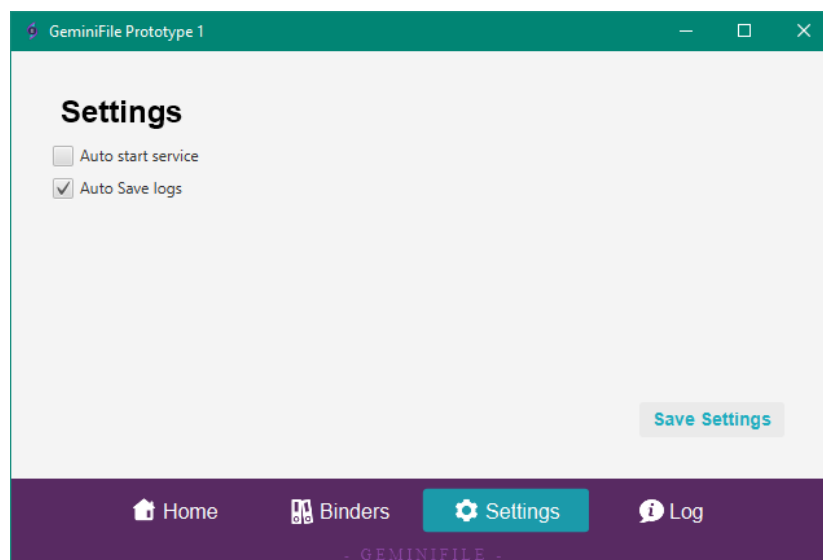


Image 3.15.4. Settings menu

The settings menu is lackluster for now but is open for more configurations in the future. The GUI service will create a new configuration file to save all the settings. When the GUI service is started, the file will be loaded, and necessary functions will be run based on the configurations. If the file does not exist, then a new default configuration file will be created.

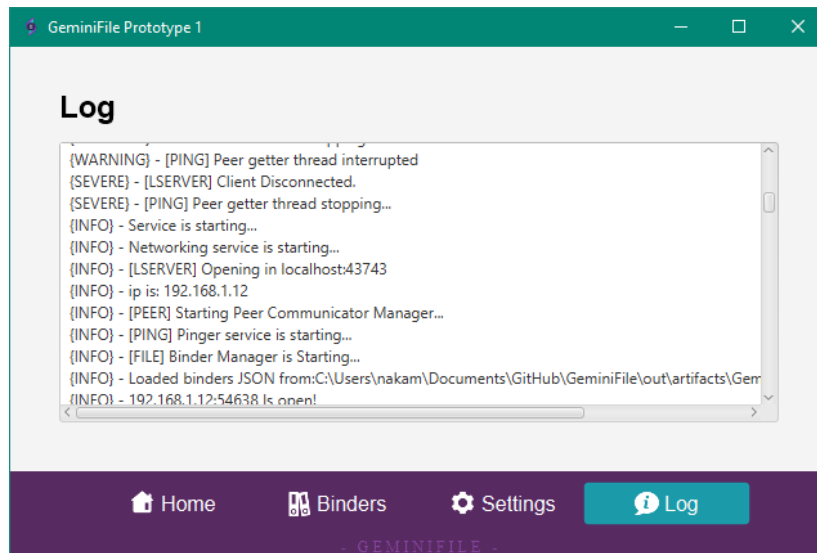


Image 3.15.5. The log page.

To keep consistency with the CLI version of GeminiFile, a log page has been dedicated to keep track of every operations done and exceptions thrown by the program. The log is displayed in a TextArea class provided by JavaFX. From all the output streams gathered from the service's logger, messages are queued then displayed on the log page.

JavaFX is a powerful API to create modern Java based GUI applications, but there is surely a learning curve to mastering it. In the current state of the application, the GUI suffices all the user needs in my opinion.

5. Working Program Evidence

To make things clear, the CLI version of GeminiFile will be shown working here. Two devices will be used in the process and communication. The left terminal (The dark blue background) is my Windows 10 laptop, running on PowerShell. The left right terminal is my home Raspberry Pi Linux server, running on Debian 10 Buster on Linux terminal.

```
PS C:\Users\nakan\Documents\GitHub\GeminiFile\out\artifacts\GeminiFile_jar> java -jar GeminiFile
start
[GeminiFile log v0.0.1]
(INFO) - Service is starting...
(INFO) - Networking service is starting...
(INFO) - [LSERVER] Opening in localhost:43743
(INFO) - ip is: 192.168.1.12
(INFO) - [PING] Pinger service is starting...
(INFO) - [PEER] Starting Peer Communicator Manager...
(INFO) - [FILE] Binder Manager is Starting...
(INFO) - Loaded binders JSON from C:\Users\nakan\Documents\GitHub\GeminiFile\out\artifacts\Gemini
File\binders.json
(INFO) - [Binder] Starting directory watcher TestBinder
(INFO) - [NetFile] Starting file receiver service...
(INFO) - Registered C:\Users\nakan\Desktop\TestSync to directory watcher
(INFO) - Registered C:\Users\nakan\Desktop\TestSync\New folder to directory watcher
(INFO) - Registered C:\Users\nakan\Desktop\TestSync\New folder\Gardyan to directory watcher
(INFO) - Registered C:\Users\nakan\Desktop\TestSync\New folder\Gardyan\New folder to directory w
atcher
(INFO) - Registered C:\Users\nakan\Desktop\TestSync\New folder\OFTheWistWrag to directory watch
er
(INFO) - Registered C:\Users\nakan\Desktop\TestSync\New folder>Please do something about this d
irectory watcher
(INFO) - 192.168.1.12:54638 is open!
(INFO) - 192.168.1.4:54638 is open!
(INFO) - [PING] Process done after: 8851
(INFO) - [PING] Process will restart after: 21141
(INFO) - [PEER] Successfully established connection from 192.168.1.4
(INFO) - [NetFile] Add binder here to 192.168.1.4
(INFO) - [PEER] Sending Message Type ASK
(INFO) - [PEER] Received Message Type ASK
(INFO) - [PEER] Sending Message Type ASK
(INFO) - [PEER] Received Message Type ASK
(INFO) - [NetFile] Starting data receive operation
(INFO) - [NetFile] Will send file: C:\Users\nakan\Desktop\TestSync\New folder\Gardyan\Help me m
ake it through the night.txt
(INFO) - [NetFile] Sent Help me make it through the night.txt with 34 Bytes in 1 block(s)
(INFO) - [NetFile] Will send file: C:\Users\nakan\Desktop\TestSync\New folder\Gardyan\IPEELODNT
HEARH.txt
(INFO) - [NetFile] Sent IPEELODNTHEARH.txt with 0 Bytes in 0 block(s)
(INFO) - [NetFile] Will send file: C:\Users\nakan\Desktop\TestSync\New folder\Gardyan\New folder
\Commy of ye hes.rar with 24 Bytes in 1 block(s)
(INFO) - [NetFile] Sent Commy of ye hes.rar with 24 Bytes in 1 block(s)
(INFO) - [NetFile] Will send file: C:\Users\nakan\Desktop\TestSync\New folder\Gardyan\unknown.pn
pi@athena:~$ java -jar GeminiFile.jar start
[GeminiFile log v0.0.1]
(INFO) - Service is starting...
(INFO) - Networking service is starting...
(INFO) - [LSERVER] Opening in localhost:43743
(INFO) - ip is: 192.168.1.4
(INFO) - [PING] Pinger service is starting...
(INFO) - [PEER] Starting Peer Communicator Manager...
(INFO) - [FILE] Binder Manager is Starting...
(INFO) - [Binder] Starting directory watcher TestYes
(INFO) - [NetFile] Starting file receiver service...
(INFO) - Registered /home/pi/TestFile to directory watcher
(INFO) - 192.168.1.4:54638 is open!
(INFO) - [PING] Process done after: 9261
(INFO) - [PING] Process will restart after: 20720
(INFO) - [PEER] Successfully accepted connection from 192.168.1.12
(INFO) - [PEER] Received Message Type ASK
(INFO) - [PEER] Sending Message Type ASK
(INFO) - [PEER] Received Message Type ASK
(INFO) - [PEER] Sending Message Type ASK
(INFO) - [NetFile] Starting data receive operation
(INFO) - [NetFile] Accepted and verified data file connection from 192.168.1.12
(INFO) - [NetFile] Expected to receive Help me make it through the night.txt with 34 Bytes
(INFO) - [NetFile] Received Help me make it through the night.txt in 1 block(s) totalling 34 By
tes
(INFO) - Registered /home/pi/TestFile/New folder to directory watcher
(INFO) - Registered /home/pi/TestFile/New folder\Gardyan to directory watcher
(INFO) - [NetFile] Expected to receive IPEELODNTHEARH.txt with 0 Bytes
(INFO) - [NetFile] Expected to receive Commy of ye hes.rar with 24 Bytes
(INFO) - [NetFile] Received Commy of ye hes.rar in 1 block(s) totalling 24 Bytes
(INFO) - Registered /home/pi/TestFile/New folder\Gardyan/New folder to directory watcher
(INFO) - Registered /home/pi/TestFile/New folder\Gardyan/New folder to directory watcher
(INFO) - [NetFile] Expected to receive unknown.png with 1511205 Bytes
(INFO) - [NetFile] Received unknown.png in 369 block(s) totalling 1511205 Bytes
(INFO) - [NetFile] Expected to receive Jojo.rtf with 33 Bytes
(INFO) - [NetFile] Received Jojo.rtf in 1 block(s) totalling 33 Bytes
(INFO) - [NetFile] Expected to receive Joobby of the Joji.txt with 12 Bytes
(INFO) - [NetFile] Received Joobby of the Joji.txt in 1 block(s) totalling 12 Bytes
(INFO) - [NetFile] Expected to receive ofTheWistWrag with 24 Bytes
(INFO) - [NetFile] Received ofTheWistWrag in 1 block(s) totalling 24 Bytes
(INFO) - [NetFile] Expected to receive FREEMEDITATION.txt with 0 Bytes
(INFO) - [NetFile] Received FREEMEDITATION.txt in 0 block(s) totalling 0 Bytes
(INFO) - Registered /home/pi/TestFile/New folder/OFTheWistWrag to directory watcher
```

Image 5.1. Overview of the connections.

As seen on the image, both the devices begin initially not connected to each other. The Windows machine runs the peer scanner service first, and discovered the Raspberry pi machine first, in IP 192.168.1.12 in port 54638. After that, the Windows machine attempts to establish connection to the other peer. Connection is then accepted by the Linux machine as seen on the right image, near the middle.

```
{INFO} - [PEER] Received Message Type ASK
{INFO} - [PEER] Sending Message Type ASK
```

Image 5.2. Back and forth messaging.

This is an example of communication with both machines. Every message sent and received will be reported on the console for the user to see its' intention.

```
{INFO} - 192.168.1.12:54638 Is open!  
{INFO} - 192.168.1.4:54638 Is open!  
{INFO} - [PING] Process done after: 8851  
{INFO} - [PING] Process will restart after: 21141  
{INFO} - [PEER] Successfully established connection with 192.168.1.4  
  
{INFO} - 192.168.1.4:54638 Is open!  
{INFO} - [PING] Process done after: 9261  
{INFO} - [PING] Process will restart after: 20720  
{INFO} - [PEER] Successfully accepted connection from 192.168.1.12
```

Image 5.3. and 5.4. Connection queries are sent and accepted by the peers involved.

As seen on image 5.3. and 5.4., the peer scanner (PING) service occasionally appears and reports what peers it has detected in the network.

```
{INFO} - Registered C:\Users\nakam\Desktop\TestSync to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\New folder to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag to directory watcher
{INFO} - Registered C:\Users\nakam\Desktop\TestSync\New folder>Please do something about this to directory watcher
```

Image 5.5. Registering all the directories to watcher.

```
{INFO} - [NetFile] Starting delta send operation
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\Help me make it trthough the night.txt
{INFO} - [NetFile] Sent Help me make it trthough the night.txt with 34 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\IFEELDOWNTHEEARTH.txt
{INFO} - [NetFile] Sent IFEELDOWNTHEEARTH.txt with 0 Bytes in 0 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\New folder\Commy of ye hes.rar
{INFO} - [NetFile] Sent Commy of ye hes.rar with 24 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Gardyan\unknown.png
{INFO} - [NetFile] Sent unknown.png with 1511205 Bytes in 369 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Jojo.rtf
{INFO} - [NetFile] Sent Jojo.rtf with 33 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Joobsy of the joji.txt
{INFO} - [NetFile] Sent Joobsy of the joji.txt with 12 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\ofTheOOF.rar
{INFO} - [NetFile] Sent ofTheOOF.rar with 24 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag\FREEMEDITATION.txt
{INFO} - [NetFile] Sent FREEMEDITATION.txt with 0 Bytes in 0 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag\Jason.zip
{INFO} - [NetFile] Sent Jason.zip with 22 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\OfTheWristWrag\JENS.txt
{INFO} - [NetFile] Sent JENS.txt with 86 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder>Please do something about this\00ooo00oo.txt
{INFO} - [NetFile] Sent 00ooo00oo.txt with 21 Bytes in 1 block(s)
{INFO} - [NetFile] Will send file: C:\Users\nakam\Desktop\TestSync\New folder\Result.png
{INFO} - [NetFile] Sent Result.png with 29384 Bytes in 8 block(s)
{INFO} - [NetFile] Completed delta send operation token SqTgTywK5G
{WARNING} - [PEER] Disconnected from: 192.168.1.4
```

```
{INFO} - [NetFile] Starting delta receive operation
{INFO} - [NetFile] Accepted and verified delta file connection from 192.168.1.12
{INFO} - [NetFile] Expected to receive Help me make it trthough the night.txt with 34 Bytes
{INFO} - [NetFile] Received Help me make it trthough the night.txt in 1 block(s) totalling 34 Bytes
{INFO} - Registered /home/pi/TestFile/New folder to directory watcher
{INFO} - Registered /home/pi/TestFile/New folder/Gardyan to directory watcher
{INFO} - [NetFile] Expected to receive IFEELDOWNTHEEARTH.txt with 0 Bytes
{INFO} - [NetFile] Received IFEELDOWNTHEEARTH.txt in 0 block(s) totalling 0 Bytes
{INFO} - [NetFile] Expected to receive Commy of ye hes.rar with 24 Bytes
{INFO} - [NetFile] Received Commy of ye hes.rar in 1 block(s) totalling 24 Bytes
{INFO} - Registered /home/pi/TestFile/New folder/Gardyan/New folder to directory watcher
{INFO} - Registered /home/pi/TestFile/New folder/Gardyan/New folder to directory watcher
{INFO} - [NetFile] Expected to receive unknown.png with 1511205 Bytes
{INFO} - [NetFile] Received unknown.png in 369 block(s) totalling 1511205 Bytes
{INFO} - [NetFile] Expected to receive Jojo.rtf with 33 Bytes
{INFO} - [NetFile] Received Jojo.rtf in 1 block(s) totalling 33 Bytes
{INFO} - [NetFile] Expected to receive Joobsy of the joji.txt with 12 Bytes
{INFO} - [NetFile] Received Joobsy of the joji.txt in 1 block(s) totalling 12 Bytes
{INFO} - [NetFile] Expected to receive ofTheOOF.rar with 24 Bytes
{INFO} - [NetFile] Received ofTheOOF.rar in 1 block(s) totalling 24 Bytes
{INFO} - [NetFile] Expected to receive FREEMEDITATION.txt with 0 Bytes
{INFO} - [NetFile] Received FREEMEDITATION.txt in 0 block(s) totalling 0 Bytes
{INFO} - Registered /home/pi/TestFile/New folder/OfTheWristWrag to directory watcher
{INFO} - Registered /home/pi/TestFile/New folder/OfTheWristWrag to directory watcher
{INFO} - [NetFile] Expected to receive Jason.zip with 22 Bytes
{INFO} - [NetFile] Received Jason.zip in 1 block(s) totalling 22 Bytes
{INFO} - [NetFile] Expected to receive JENS.txt with 86 Bytes
{INFO} - [NetFile] Received JENS.txt in 1 block(s) totalling 86 Bytes
{INFO} - [NetFile] Expected to receive 00ooo00oo.txt with 21 Bytes
{INFO} - [NetFile] Received 00ooo00oo.txt in 1 block(s) totalling 21 Bytes
{INFO} - Registered /home/pi/TestFile/New folder>Please do something about this to directory watcher
{INFO} - Registered /home/pi/TestFile/New folder>Please do something about this to directory watcher
{INFO} - [NetFile] Expected to receive Result.png with 29384 Bytes
{INFO} - [NetFile] Received Result.png in 8 block(s) totalling 29384 Bytes
{INFO} - [NetFile] Completed delta receive operation token SqTgTywK5G
```

Image 5.6. and 5.7. File transfer operations undertaken by both the peers.

The GeminiFile instance on the Windows machine detects several files missing from the Linux server. Therefore, it arranges a communication to the Linux machine to sync those missing files. After negotiating, the operation is approved by both machines, and the windows machine starts to send all the file necessary o the Linux machine.

```
{INFO} - [Binder] 3 Change detected in binder 'TestBinder'. Will query to other peers.  
{INFO} - Will send files:  
{INFO} - \New folder\OfTheWristWrag\IMG_20200303_110921.jpg  
{INFO} - \New folder\OfTheWristWrag\girl-3840x2160-bicycle-city-4k-19777.jpg  
{INFO} - \New folder\OfTheWristWrag\grblbatik.jpg  
{INFO} - [PEER] Sending Message Type ASK  
{INFO} - [PEER] Received Message Type ASK  
{INFO} - [NetFile] Starting delta send operation
```

Image 5.8. The service has detected a new change in a folder.

In this example, the service detects several changes in the folder TestBinder. The service will wait some more time, to make sure there isn't any other modification and new files in the folder, before sending a query to other devices that shares the same folder id.

6. Closing Remarks

The program has been finished and is currently usable in its current form. However, because this is my first time writing a program of this magnitude, and because I am still in education, the code is very far from perfect. In the end, a lot of experience has been gathered from these past 2 months coding this application, and I am satisfied that I was given the opportunity to learn this.

For the future, I do have a lot of ideas to make this application real, and production ready. I have been taking notes all the ideas occurred since the start I am writing this report. A lot of fixes, better implementation, and new features in the application. However, I do not know myself whether I want to continue this project or delve into another, bigger and diverse projects. Other than Java.

7. Resources and References

¹ <https://github.com/stleary/JSON-java>
<https://openjfx.io/>

Peer to Peer: <http://cs.berry.edu/~nhamid/p2p/>

Great java Reference, well-structured and uses easy and understandable language.
Recommend: <https://www.codejava.net/>

Documentation: <https://docs.oracle.com/en/java/>

IDE: <https://www.jetbrains.com/idea/>

UML Generators:

- PlantUML integration: <https://plugins.jetbrains.com/plugin/7017-plantuml-integration>
- PlantUML generator: <https://plugins.jetbrains.com/plugin/10387-sketch-it->
- Visualization tool: <https://graphviz.org/>

Scene Builder: <https://gluonhq.com/products/scene-builder/>

Great and beautiful git manager: <https://www.gitkraken.com/>