Distributed Systems

Steganogram Project

Project Contributors:

Aley El-Din Baracat - 900140290

John H. Sourour - 900140842

Sara Seleem - 900140907

Yasmin ElDokany - 900131538

CSCE 445/4411

Prof. Amr El-Kadi

Fall 2017

Table of Contents

1. System Overview

This system implements an image sharing application, using UNIX sockets and UDP to implement Remote Invocation Middleware (RMI) and grant peer users the capabilities to control their images' sharing privileges.

The system has been fully implemented and tested on machines running the Ubuntu 16.04 Linux OS, using C++11.

2. System Dependencies

The system has the following dependencies:

- **Steghide**

  The system uses the Steghide tool; a tool that provides steganography program capable of hiding data in various kinds of images (most importantly .JPEG, .JPG and .PNG images).

  To install this tool on Ubuntu, use the command:

  ```
  sudo apt-get install steghide
  ```

3. Execution Guide

To execute the system's code, there should be a machine running the system's service command line interface and user machines running the Qt application developed for it.

4. System Architecture

The system's structure is object-oriented, composed of the following classes (▼ illustrates abstract classes):

▼ Data

- **AckData**
- **AuthData**

- **ImageData**
- **ImageListData**
- **ImageRequestData**
- **PingData**
- **StatusData**
- **ViewsReplyData**
- **ViewsRequestData**

- Message

- PackGen

- Peer

- Service

- UDPSocket

▼ QDialog

- **ImageList**

▼ QMainWindow

- **imageviewer**
- **LoginWindow**
- **UserWindow**

4.1. Data Class

This class, as illustrated in **figure 4.1**, is an abstract class that represents data objects that can be embedded in the system's messages to allow peers to communicate data over the network.

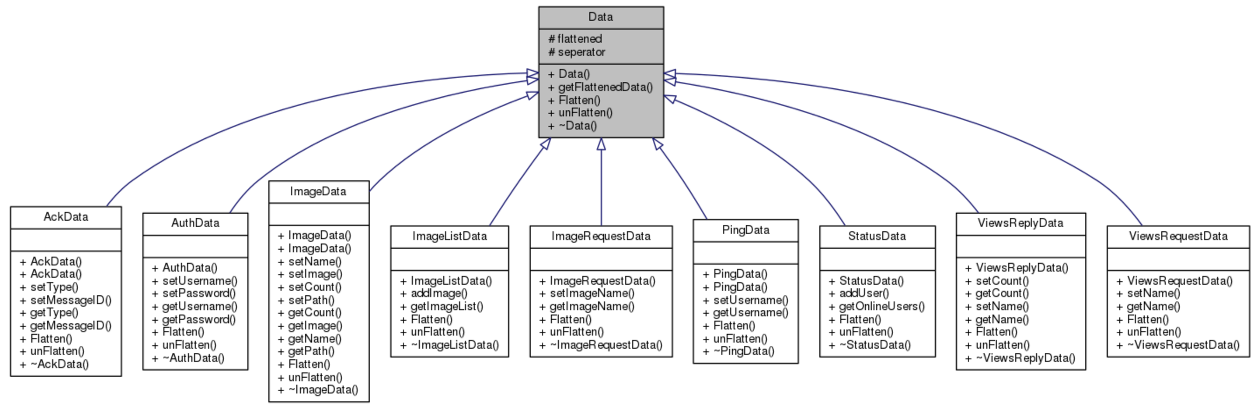


Figure 4.1 | Data Class Hierarchy

It has the following public member functions:

- **Data ():** An abstract data object constructed implemented in the inherited data classes.
- **Flatten ():** An abstract function that returns a boolean variable indicating successful data flattening, i.e marshelling, into a string object stored in the **flattened** string mentioned below, implemented in the respective inheirted data classes.
- **getFlattenedData ():** A function that returns the **flattened** data string.
- **unFlatten (string s):** An abstract function that returns a boolean variable indicating unflattens a string into the respective data type, based on the inherited data object on whom the function is called.
- **~Data ():** A data object destructor.

And it has the following protected attributes:

- **string flattened:** A string variable that stores the flattened format of the ata object.
- **const char seperator:** A constant string variable

The inherited data object types are implemented by the following classes.

4.1.1. ImageData Class

Data class inherited object that defines image data and implements their respective functionalities.

4.1.2. AckData Class

Data class inherited object that defines acknowlegment data and implements their respective functionalities.

4.1.3. PingData Class

Data class inherited object that defines ping messages' data content and implements their respective functionalities.

4.1.4. ViewsReplyData Class

Data class inherited object that defines extra image views request reply's data and implements their respective functionalities.

4.1.5. AuthData Class

Data class inherited object that defines user authentication data and implements their respective functionalities.

4.1.6. StatusData Class

Data class inherited object that defines currently online user list's data and implements their respective functionalities.

4.1.7. ImageListData Class

Data class inherited object that defines a user image list’s data and implements their respective functionalities.

4.1.8. ViewsRequestData Class

Data class inherited object that defines extra image views request reply’s data and implements their respective functionalities.

4.1.9. ImageRequestData Class

Data class inherited object that defines image requests’ data and implements their respective functionalities.

4.2. Message Class

This class is responsible for the packaging of the messages communicated over the network.

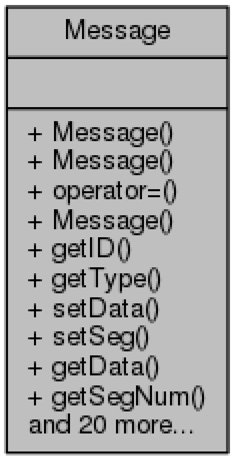


Figure 4.2 | Message Class

It has the following public member functions:

- **Message ():** Message object default constructor
- **Message (const Message &other):** Message object copy constructor
- **Message& operator= (const Message &other):** Message object copy assignment constructor
- **Message (MessageType type, string IP, int Port, string targetIP, int targetPort):** Message object constructor that takes the message's type, which must be one of the following types:
    - Ping
    - Auth
    - StatusReply
    - ImageListReply
    - ImageReply
    - Ack
    - NegAck
    - StatusRequest
    - ImageListRequest
    - ViewsRequest
    - DenyRequest
    - ImageRequest
    - ViewsReply
    - Terminate

  And it takes the message's owner's IP address and server port number (for acknowledgement replies), and the message target's IP address and server port number.

- **string getID ():** Function that returns a unique message ID that combines a static count of a single user's messags and their username.
- **MessageType getType ():** Function that returns a message's type.
- **bool setData (Data &d):** Function that sets the message's data object field.
- **void setSeg (int total, int current):** Function that sets the message's segment sequence number.
- **string getData ():** Function that gets a message object's flattened data object string.
- **int getSegNum ():** Function that returns the message object's segment's sequence number.

- **int getSegTot ():** Function that returns the total number of segments of this message object.
- **int getOwnerPort ():** Fuction that returns a message owner's server's port number.
- **string getOwnerIP ():** Fuction that returns a message owner's server's IP address.
- **int getTargetPort ():** Fuction that returns a message target's server's port number
- **string getTargetIP ():** Fuction that returns a message target's server's IP address.
- **int getDataSize ():** Function that returns the message object's data's size.
- **int getTotalSize ():** Function that returns the message object's total size.
- **void setFlattenedData (string s): :** Function that sets the message's data object's flattened data string.
- **void setMessageID (string num):** Function that returns the message ID of a message object.
- **void setOwnerPort (int num):** Function that sets the message owner's server's port.
- **void setOwnerIP (string s):** Function that sets the message owner's IP address.
- **void setTargetPort (int num):** Function that sets the message target's server's port.
- **void setTargetIP (string s):** Function that sets the message target's IP address.
- **void setType (MessageType mt):** Function that sets the type of a message.
- **void setDataSize (int i):** Function that sets the data size of a message.
- **bool Flatten ():** Function that returns a boolean variable based on the successful flattening of a message object.
- **bool unFlatten (string s):** Function that returns a boolean variable based on the successful unflattening of a string message.
- **string getFlattenedMessage ():** Function that returns a message's flattened string format.
- **void printMessageDetails ():** Function that prints a message's details.
- **~Message ():** Message object destructor.

4.3. PackGen Class

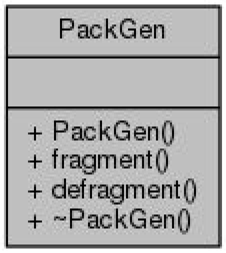


Figure 4.3 | Packgen Class

This class is responsible for the generation of packets to be sent over sockets, and is responsible for the fragmentation and defragmentation of large message objects (especially those including ImageData objects of large size) into segments of size 50 Kbyes, which is less than the 64k bytes threshold handled by UDP sockets.

4.4. Peer Class

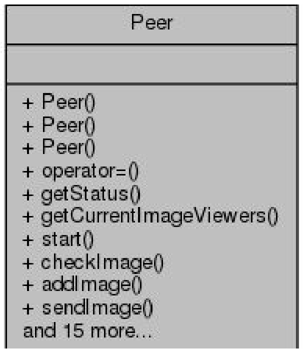


Figure 4.4 | Peer Class

This class is responsible for the client/server simultaneous functionalities of a user of the system. It includes 20 synchronized worker threads divided evenly to handle message exchange services for the peer users.

It has the following public member functions:

- **Peer ()**
- **Peer (char *_listen_hostname, int _listen_port, char *service_hostname, int service_port)**

- **Peer (const Peer &other)**
- **Peer & operator= (const Peer &other)**
- **std::map< string, pair< string, int > > getStatus ()**
- **vector< pair< string, int > > getCurrentImageViewers (string)**
- **void start ()**
- **int checkImage (string name, string ip)**
- **void addImage (string name, string path)**
- **void sendImage (string name, string IP, int port, int count)**
- **std::set< string > getMyImages ()**
- **std::set< string > getUserImages (string username)**
- **void setLocalImages (std::set< string >)**
- **string getMBoxTitle ()**
- **string getMBoxRequest ()**
- **void setMBoxBool (bool)**
- **bool login (string username, string password)**
- **void execute (Message msg)**
- **void requestImage (string name, string user)**
- **void logOff ()**
- **void requestViews (string name, string user)**
- **void addViews (int count, string image, string user)**
- **void halt ()**
- **void processReply ()**
- **~Peer ()**

It has the following Qt window signals:

- **void terminateProgram ()**
- **void firstWindow ()**

4.5. Service Class

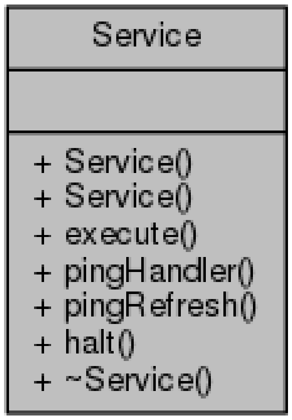


Figure 4.5 | Service Class

This class is responsible for the Steganogram system's services to its users, which includes the maintenance of its user directory, the refreshing of its online user's list, and the proper termination of its services. It includes 20 synchronized worker threads divided evenly to handle message exchange services with its peer users.

4.6. UDPSocket Class

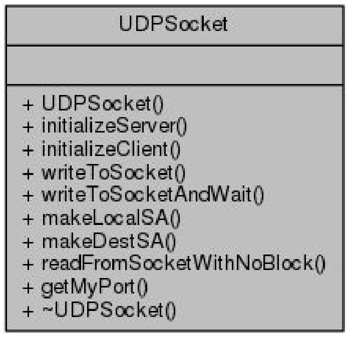


Figure 4.6 | UDPSocket Class

This class is responsible for the socket management and handling through the sending and receiving processes of user and service messages.

4.7. Qt Window Classes

These classes include:

- **ImageList Class**
- **Imageviewer Class**
- **LoginWindow Class**
- **UserWindow Class**

They are responsible for the system's graphical user interface (GUI) handling and will be explained further below through their interfaces' properties.

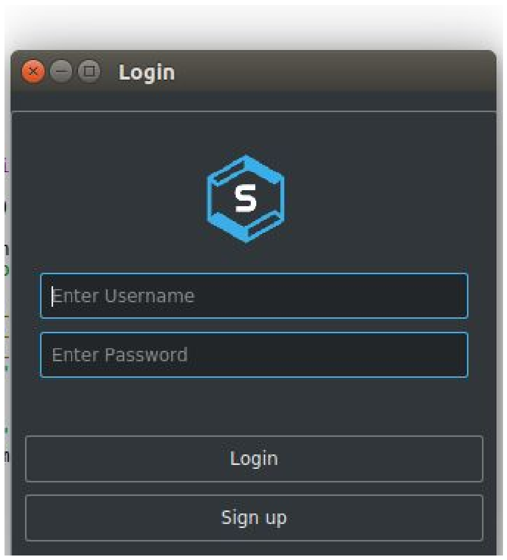
5. Graphical User Interface Guide

The Steganogram system's GUI has been built using Qt Creator 4.0.2.

5.1. Login Window

On starting up the Steganogram user application, the user is prompted to log in, as shown in **figure 5.1**. The login process can have four outcomes:

- Successful login which will grant the user access to the home Steganogram window of the application, shown in **figure 5.5.**
- Unsuccessful login due to invalid user credentials, which will display a prompt indicating an unsuccessful login attempt, as shown in **figure 5.2**.
- Unsuccessful login due to the server being disconnected, which displays a pop-up indicating that, as shown in **figure 5.3**.
- Unsuccessful login due to a user being logged in with these credentials on a different machine, which will display a prompt indicating an unsuccessful login attempt, as shown in **figure 5.4**.



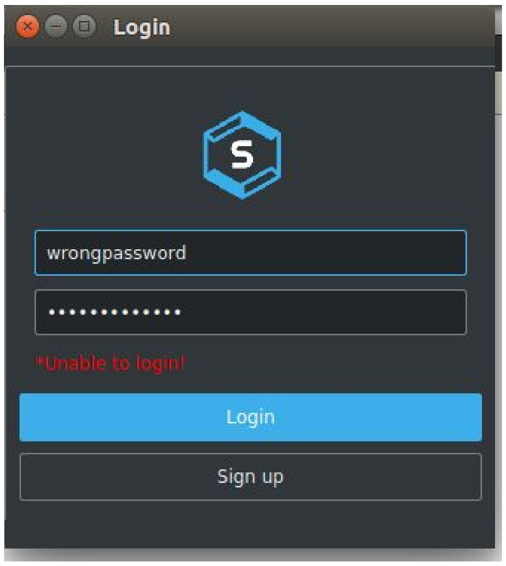


Figure 5.1 | Initial Login Window

Figure 5.2 | Invalid User Login Window

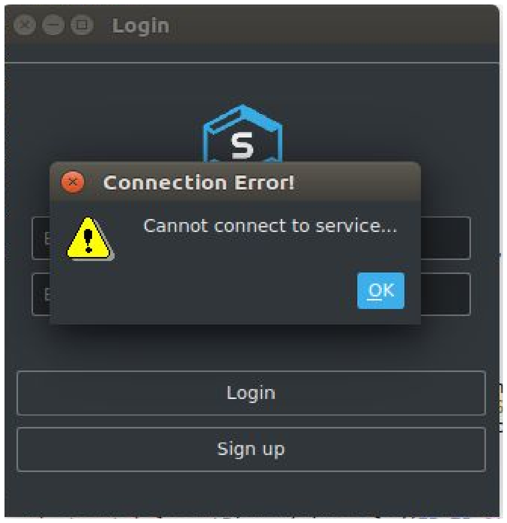
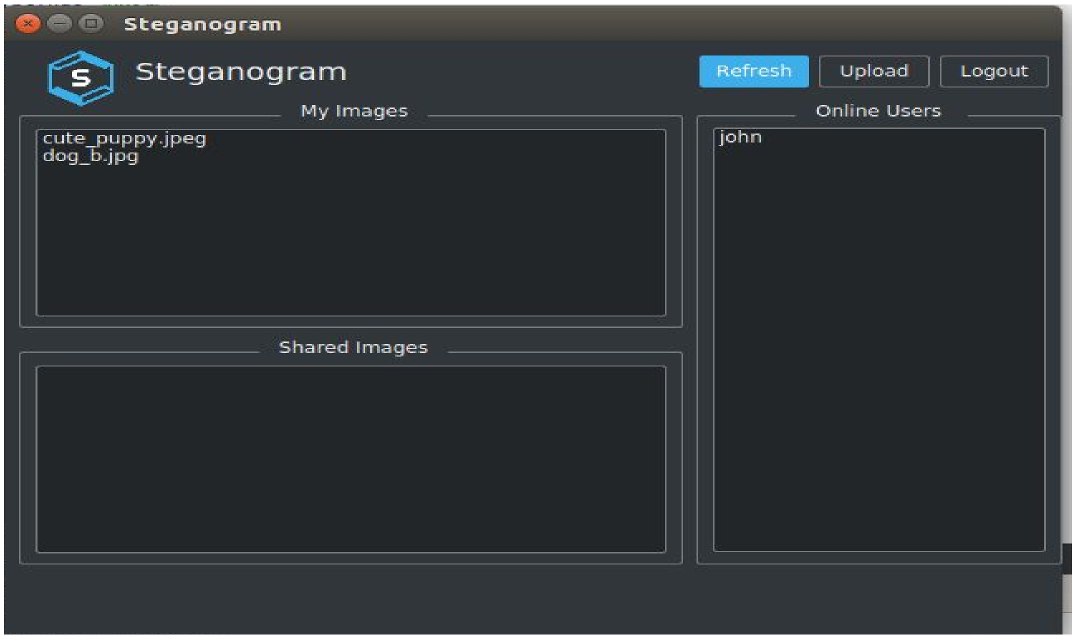


Figure 5.3 | Disconnected Service Error

Figure 5.4 | User Session Running Elsewhere

5.2. Home Window

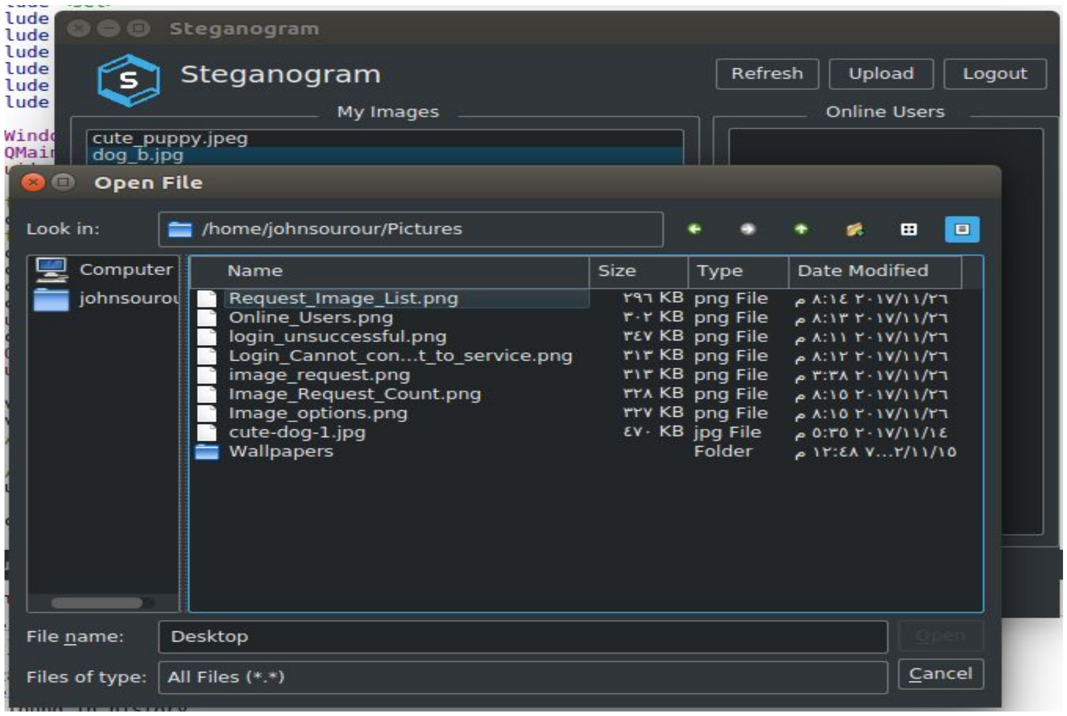
The home Steganogram window contains the user's images available on their machine.



cute_puppy.jpeg
dog_b.jpg

john

Figure 5.5 | Online users are shown on the right



cute_puppy.jpeg
dog_b.jpg

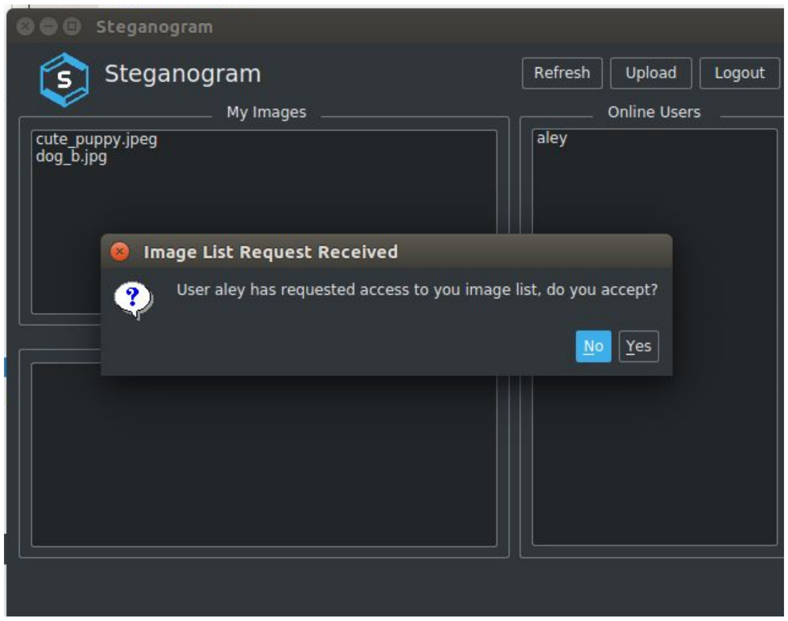
Look in: /home/johnsourour/Pictures

Name	Size	Type	Date Modified
Request_Image_List.png	٢٩٦ KB	png File	م ٨:١٤ ٢٠١٧/١١/٢٦
Online_Users.png	٣٠٢ KB	png File	م ٨:١٣ ٢٠١٧/١١/٢٦
login_unsuccessful.png	٣٤٧ KB	png File	م ٨:١١ ٢٠١٧/١١/٢٦
Login_Cannot_con...t_to_service.png	٣١٣ KB	png File	م ٨:١٢ ٢٠١٧/١١/٢٦
image_request.png	٣١٣ KB	png File	م ٣:٣٨ ٢٠١٧/١١/٢٦
Image_Request_Count.png	٣٢٨ KB	png File	م ٨:١٥ ٢٠١٧/١١/٢٦
Image_options.png	٣٢٧ KB	png File	م ٨:١٥ ٢٠١٧/١١/٢٦
cute-dog-1.jpg	٤٧٠ KB	jpg File	م ٥:٣٥ ٢٠١٧/١١/١٤
Wallpapers		Folder	م ١٢:٤٨ ٧...٢/١١/١٥

File name: Desktop

Files of type: All Files (*.*)

Figure 5.6 | Upload image

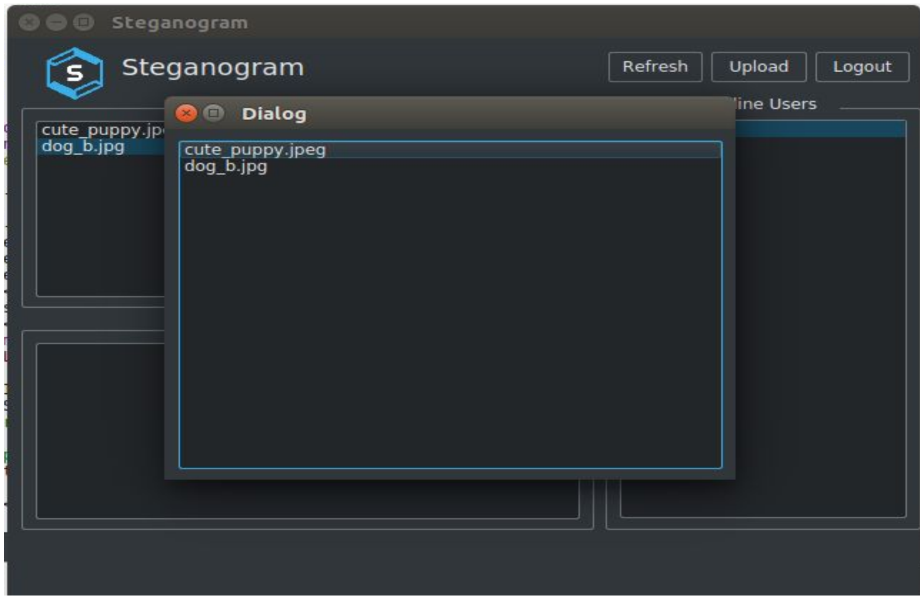


cute_puppy.jpeg
dog_b.jpg

aley

User aley has requested access to you image list, do you accept?

Figure 5.7 | Request image list



cute_puppy.jpeg
dog_b.jpg

Figure 5.8 | Image list

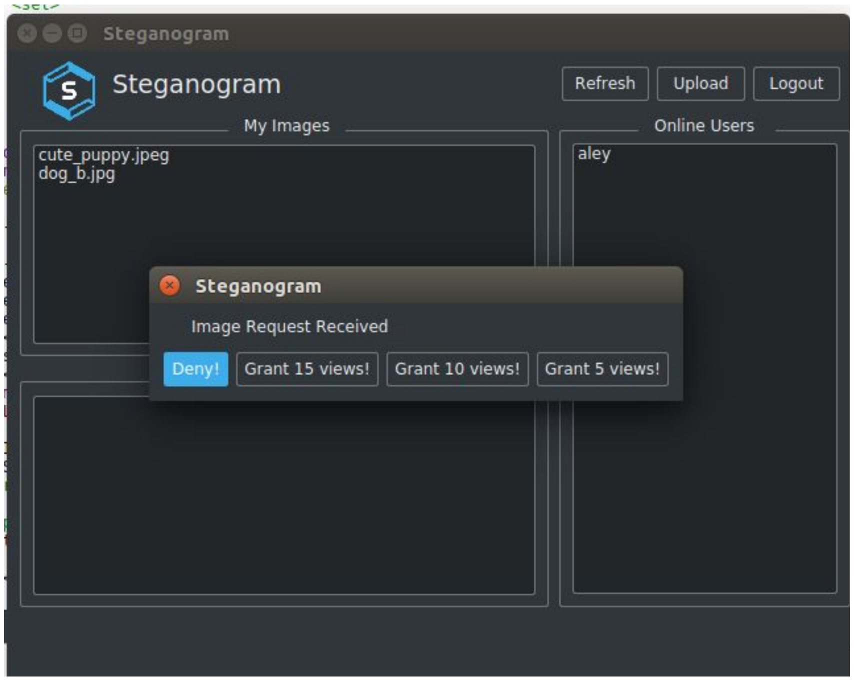


Figure 5.9 | Image request count

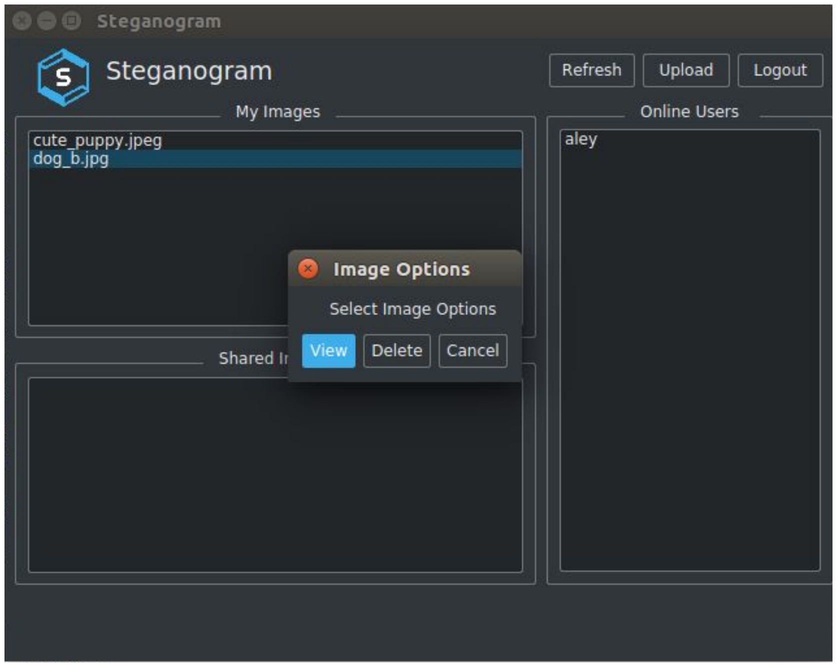
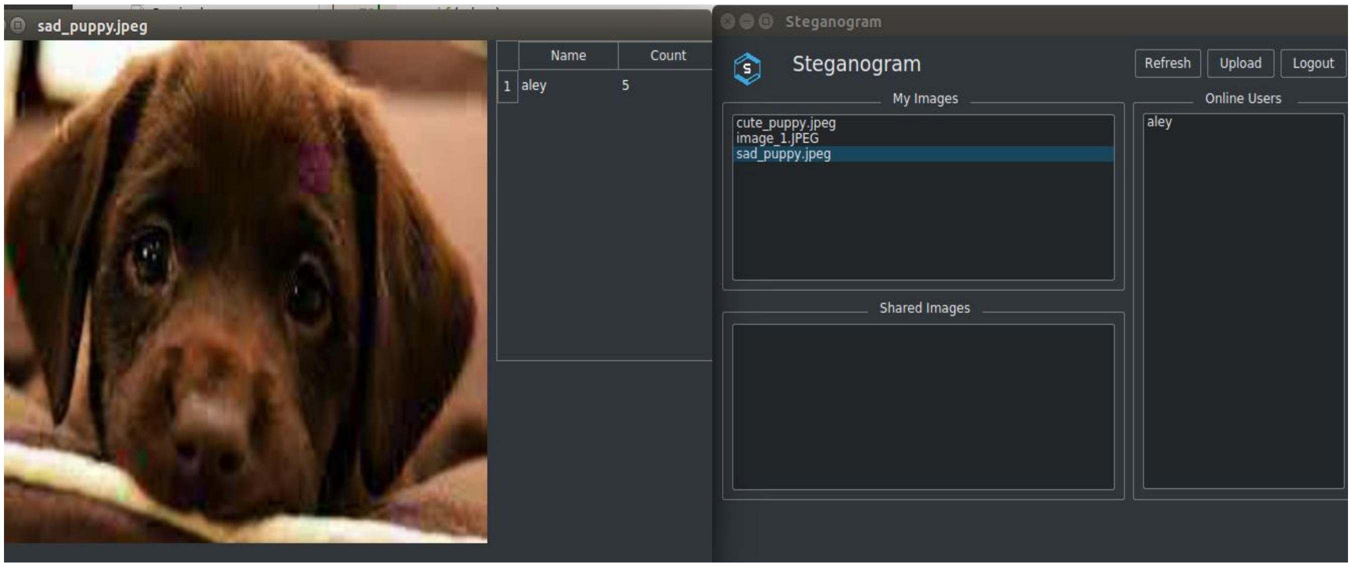


Figure 5.10 | Image options

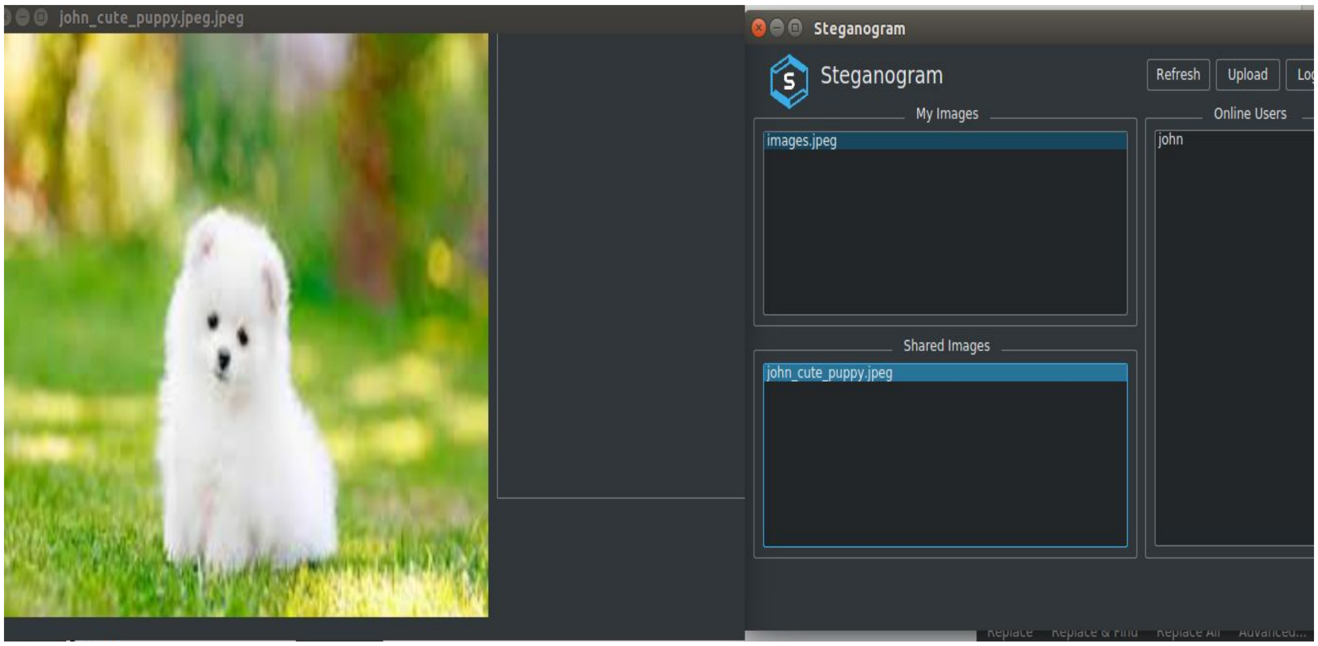


	Name	Count
1	aley	5

cute_puppy.jpeg
image_1.JPEG
sad_puppy.jpeg

aley

Figure 5.11 I Access list is shown on the left with the number of counts given to each user

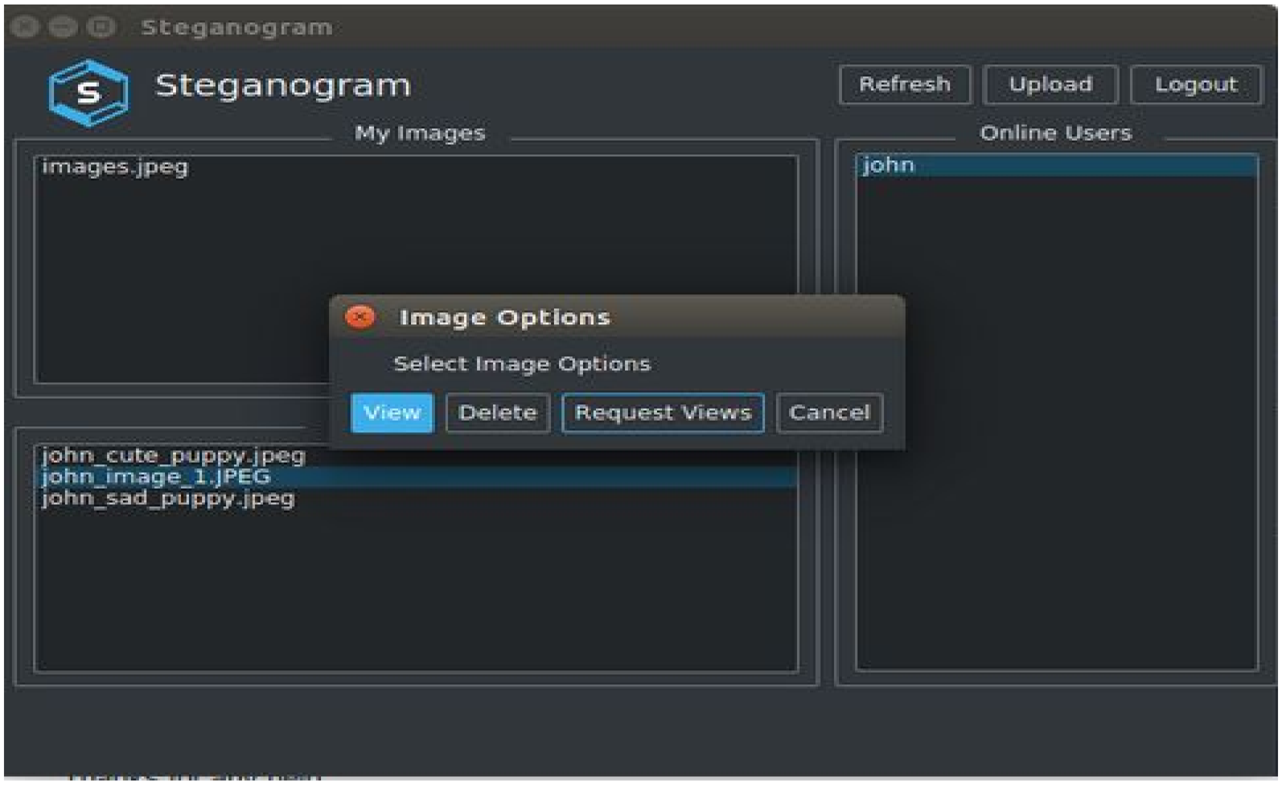


images.jpeg

john

john_cute_puppy.jpeg

Figure 5.12 I View shared image



images.jpeg

john

john_cute_puppy.jpeg
john_image_1.JPEG
john_sad_puppy.jpeg

Figure 5.13 | Request more views