

DistImage: Project Report

Group E

Fundamentals of Distributed Systems

By

Ahmed Refaay

Hassan Kandil

Lotfy Hussein

Nour Ghaly

Presented To:

Dr. Amr ElKadi

GTA Amr Saeed

1. Introduction

We are providing a peer to peer image communication system that is built on top of rpc model, RMI. using UDP socket. The protocol that we have used is a Request Reply Protocol due to the compatibility of the project.

Typically a user should be able to sign up in order to gain access to our system, login and logout. The main functionalities are:

- Viewing online and offline peers.
- Requesting an image from a chosen online peer with a certain number of views.
- Viewing an image (with the number of views decided by the owner).
- Requesting more views for the image.
- The owner of the image has the right to update the user's number of views (including revoking the viewer access to the image by setting the views to zero).
- The owner can collect statistics about the remaining views for images which he shared with the user.

2. Data Structures and Implementation Details

Directory of Service

The Directory of service (which is also known as central server) main purpose is to always be up and running so that everyone that is trying to use the system would find it available at any given point. One of its main functionalities is to handle sign up and login as well as to handle administrative services.

Main data structures:

- `struct` data;

this is a struct which has all the information that has to do with the user. Such as whether they are online or not and if so then there will be an associated current IP address and a port number. It also has a vector of all the images the user owns.

- `map<string, string>` auth_map

this is a map which has all the signed-up users and their passwords.

- `map<string, data>` users_map;

This is a map which has an empty entry for each user that is signed up. It is then updated through the functions below, which update the appropriate fields in the data struct that is mentioned above.

- `fstream` auth;

This is an in out file which replicates the auth_map, in case the server suddenly shuts down.

· `fstream` users;

This is an in out file which has the users and all their images (which are also found in the users_map). If the server suddenly shut down all the images owned by a user will still be saved however whether they are online or not will be lost and will be reset of offline.

Main functions of the Directory of service:

1. void read_first_time_users(fstream & users)

This function reads the users file line by line and files the users_map, in order to read from the map straight away when the server is up and running because it is a lot faster than reading from the file.

2. void read_first_time_auth(fstream & auth)

Similar to the read_first_time_users function, this function reads from the auth file and fills the auth_map for the same reason.

3. int sign_up (string username, string password, fstream & auth, fstream & users)

Firstly, this functions checks that no existing user has the same username as that this is passed in the function. If it does not, then it adds a field to the auth_map and to the users_map. As well as updating both the auth and users files with the required data.

4. int login (string username, string password, string current_ip, string port_num, fstream & auth)

5. void logout (string username)

6. string view ()

7. int upload (string username, string img_name, fstream & users)

Peer to Peer Communication:

Each peer has three threads: The first one handles the GUI. The second one, continuously listens for incoming requests, and for each request it opens a third thread to handle requests such as requesting an image by sending the image.

Peer Class:

Peer():

It opens two files:

- 1- imgfile: the user's images that are stored locally
 - 2- Myimagefile: the images that are shared with the user, also stored locally.
- And makes instances of the need sockets.

void listenPeer():

The peer keeps listening indefinitely for request as long as the get function is returning one. It is blocking until it receives a request.

bool getRequest():

it receives a marshalled request message. The function then unmarshals the message and extracts the op_id_request. A switch case is then done on the op_id_request, upon which one of four actions will be taken.

1. view request: the username, image name and number of views is then extracted from the unmarshaled request message. It then sends a reply to the request message sender. And views the image to the request receiver.
2. Receive image: receives an image from a user and puts the image name and the owner name under the list of images that the user can view. It then marshals the request reply and send it to the sender.
3. notify request: the username, image name and number of views is then extracted from the unmarshaled request message.
4. Request new number of views: sends a new number of users that is requested from the viewer to the image owner.

void sendReply(char *temp_buffer):

sends the marshaled reply message to the sender of the request through the

static void makeDestSA(struct sockaddr_in *sa, char *hostname, int port):

creates a socket using the ip address and port number.

int sign_up(string username, string password) :

It returns an error if there are special characters in the username or password because they are used as delimiters when sending messages. It checks that the username does not exist before. If not, it adds the username and password to the

file of all users.

int login(string username, string password):

We check that the username and password exist in our file

Time out is implemented, if the nothing is entered in the specified time, the login request is sent again up to a specific amount of time.

int logout():

Logs out the current user and clears out their port number and ip address.

Similar, to login, timeout is implemented

int upload(string imagepath):

The image path is passed to the function, if the image is found the image and its attributes are

int notify_views_by_viewer(string owner, string selectedImage, int NewNoViews):

This function notifies the owner of an image of the new number of views each time a viewer views the owner's image (the number of views get decremented each time the viewer opens an image that has been shared with him/her)

int update_views_by_owner(string viewer, string selectedImage, int NewNoViews):

This function enforces the number of views that a certain user can view the image. This directly updates the viewer's number of views.

int update_views_request_by_viewer(string owner, string selectedImage, int NewNoViews):

A viewer can request from an owner of an image to view the image more times. Upon this request the owner can choose to accept the request and therefore the number of views will be updated for the user or the request will be declined and therefore the number of views will remain the same.

int request_image(string selectedUser, string selectedImage, int views):

A viewer can request an image from an owner, by sending the selected image to the correct owner and the number of views he/she wish to view the image. The

selected owner is then notified that a user has requested an image of theirs.

void send_image(string viewerName, string selectedImage, int nViews):

Upon the request of an image, the owner of an image can either accept or decline the request. If the request is accepted then the image is sent to the viewer with the number of views that the owner has specified. This image will then be found in the file that has all the images that is shared with this specific user.

map<string, vector<string>> getUsers():

Gets all the users from the DoS file and all their attributes which includes all their images, whether they are online or not, their ip address (only if online) and their port number(only if online).

bool newimg(string img, string owner, int views):

Inserts a new entry in the images map for each image that has been shared with this specific user as well as the remaining number of views he/she has each image.

bool updateimg(string img, string owner, int views):

Update the number of views of a certain image. (writes directly to the map). This function is called if an image owner chooses to update the number of views to a certain viewer .

void readfile():

This function is called only once when the user logs in, it reads the file which has all the images that are shared with a specific user and fill it's attributes in a map. This will make writing and reading easier than reading and writing to the file each time.

void updatefile():

This function is called only once when the user logs out, to update the images file with all the changes that have been done to the map while the user was online, so that when the user logs in again he/she will have all the images and the number of views of each image updated to where they have left it.

Exercise 2 Experiment:

_____RPC_id represents an important attribute of the Message class since it allows the system to distinguish between different requests. This id starts at 0 and with each request the peer sends it gets incremented by 1 and then the peer receiving this requests unmarshals it and then copies the rpc_id of the request to the reply message (which acts as an acknowledgement to the request). Then, the requesting peer receives the reply message, unmarshals it and then checks the rpc_id of the reply to make sure it matches the one of the request it sent. In case, they don't match this means that the reply got lost (or the request didn't reach the other peer) and it needs to resend the request (since it didn't receive the correct acknowledgement).

We have implemented this in our system as shown in the screenshot below:

```
// unmarshal the reply
Message reply_view_request(reinterpret_cast<char *>(little_buffer));

string msg_view_request_reply =
    reply_view_request.getUnmarshalledMessage();

// resend if lost
int rpc_id_new = reply_view_request.getRPCId();

if (rpc_id_new != requestID) // check if the request matches the reply
{
    // if not resend the request
    int i = request_image(selectedUser, selectedImage, views);
}else
{
    cout << "msg_received_length " << msg_view_request_reply << endl;
    if (msg_view_request_reply[0] == '1')
        return 1;
    else if (msg_view_request_reply[0] == '0')
        return 0;
    else
        return 2;
}
}
```

We have tested this implementation by sending two different requests to the same peer at the same time and we made sure that the replies don't get mixed.

Exercise 3 Experiment:

Fault-tolerance capabilities of our system

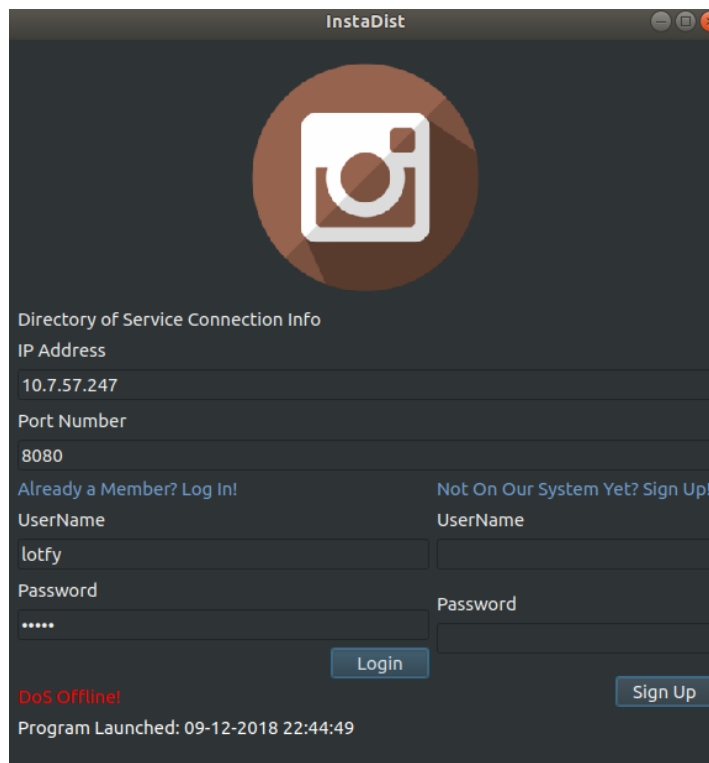
To test our system's tolerance, three scenarios are addressed:

1- When the Directory of Service is down:

When the DoS is down, any service requiring connection to the DoS will be disabled, and an error message will be displayed to the user “DoS is Offline!”

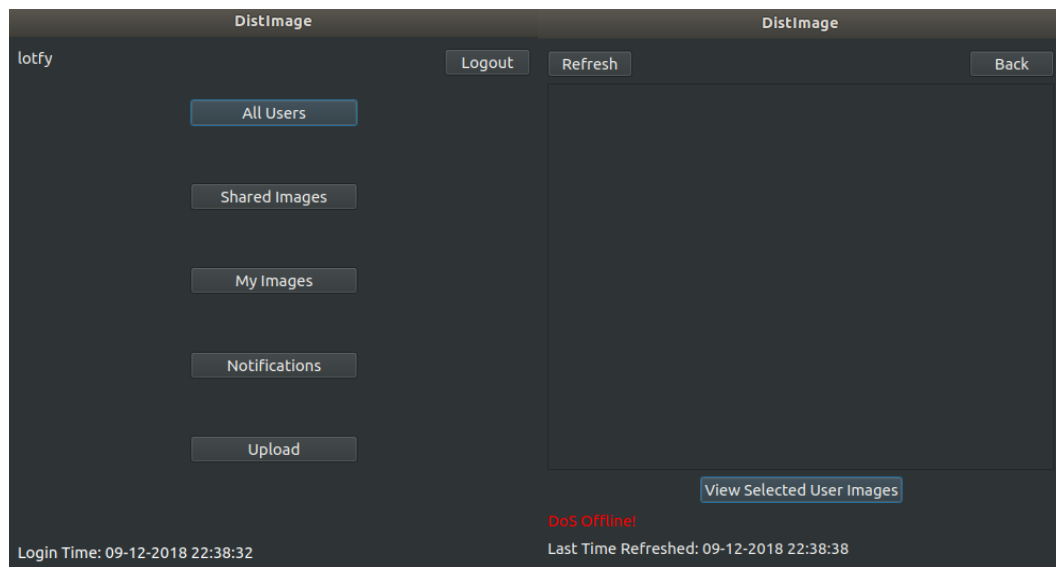
Screenshots:

A- When the user tries to login or sign up and DoS offline, an error message is shown



B- When All Users Button is pressed and DoS is offline, an error message is shown

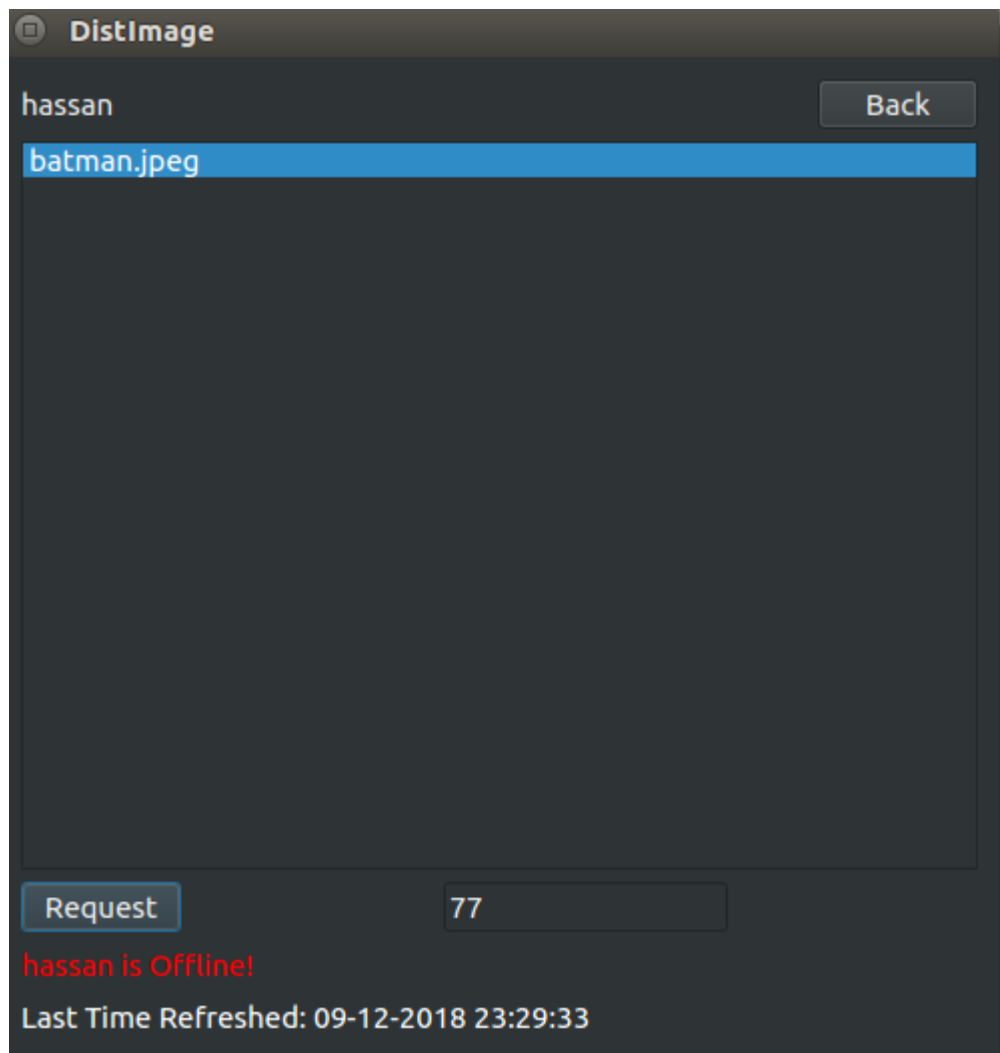
Users Button is



C- When the User tries to upload an image and DoS is offline, an error message is shown.

2- When the other peer is offline:

When a peer tries to send a request to an offline peer, an error message is shown.

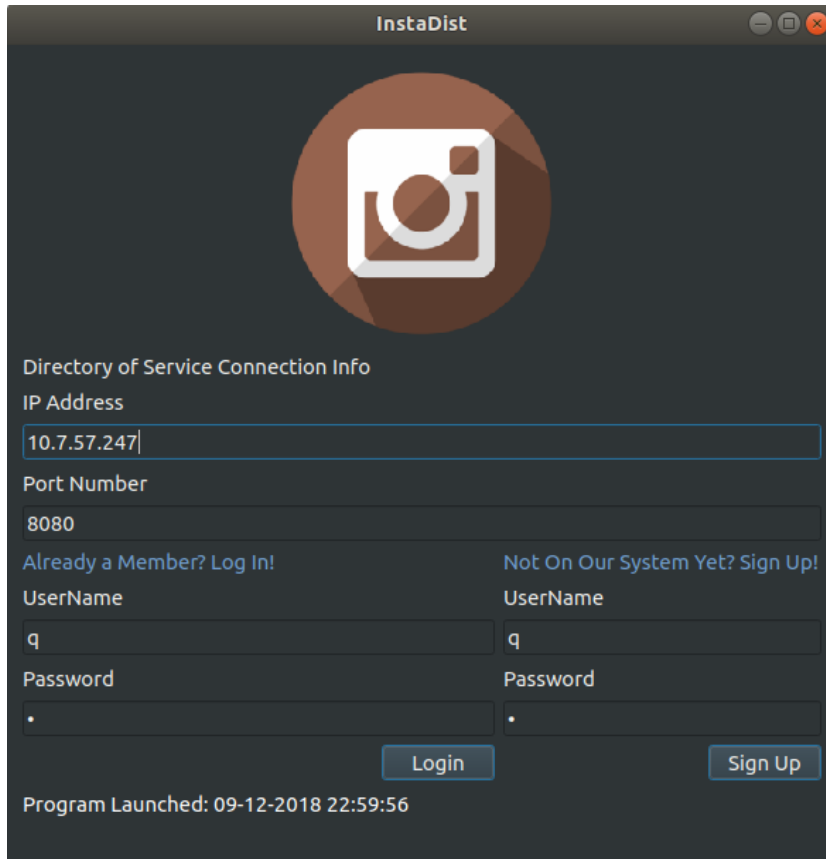


3- When there is no internet connection:


An error message is shown saying "No Internet Connection."

Screenshots:

Login and sign up screen



InstaDist



Directory of Service Connection Info

IP Address

10.7.57.247

Port Number

8080

Already a Member? Log In!

Not On Our System Yet? Sign Up!

UserName

UserName

q

q

Password

Password

•

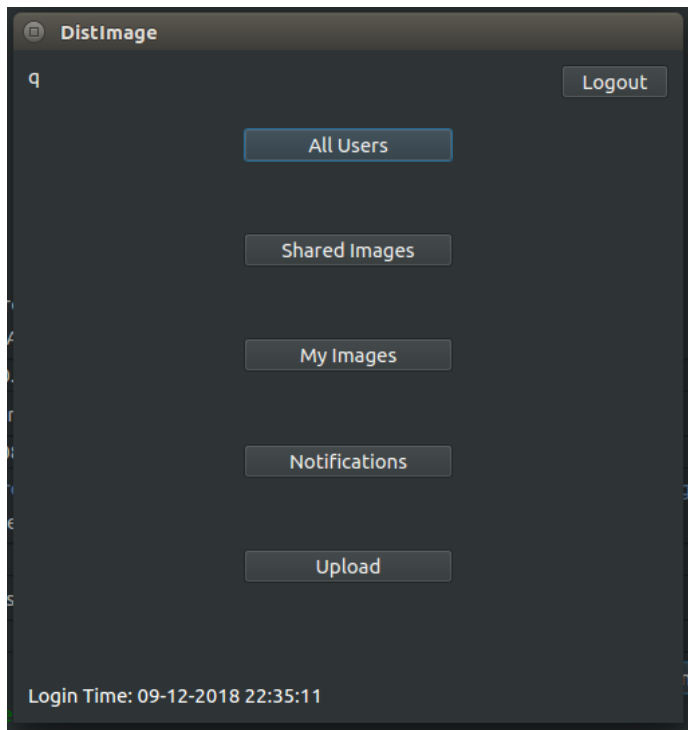
•

Login

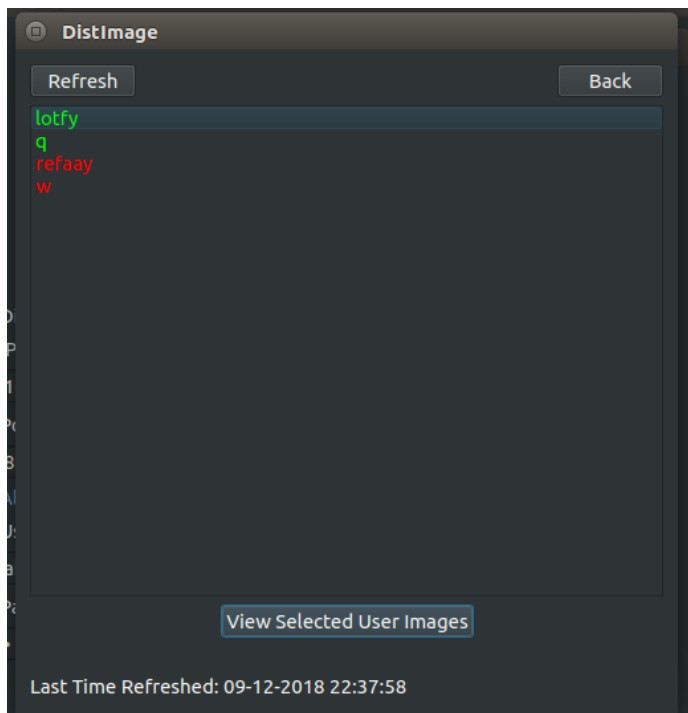
Sign Up

Program Launched: 09-12-2018 22:59:56

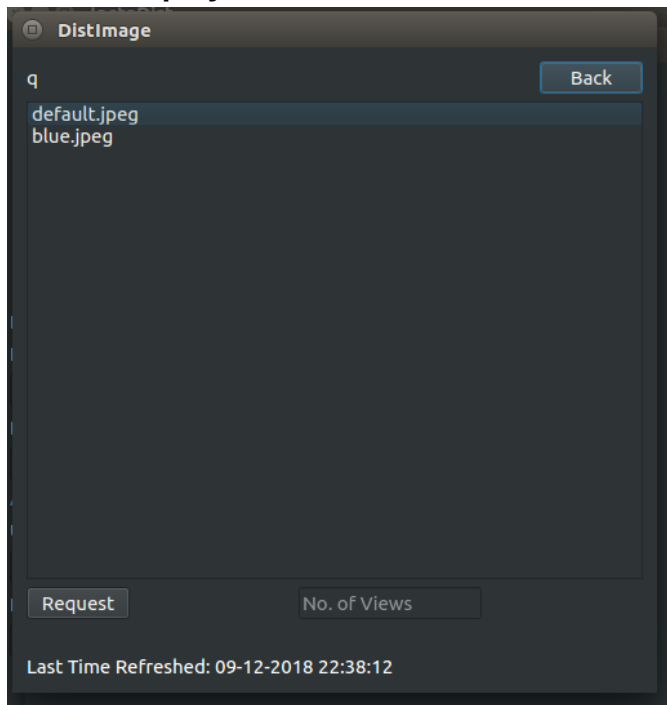
First screen the user sees



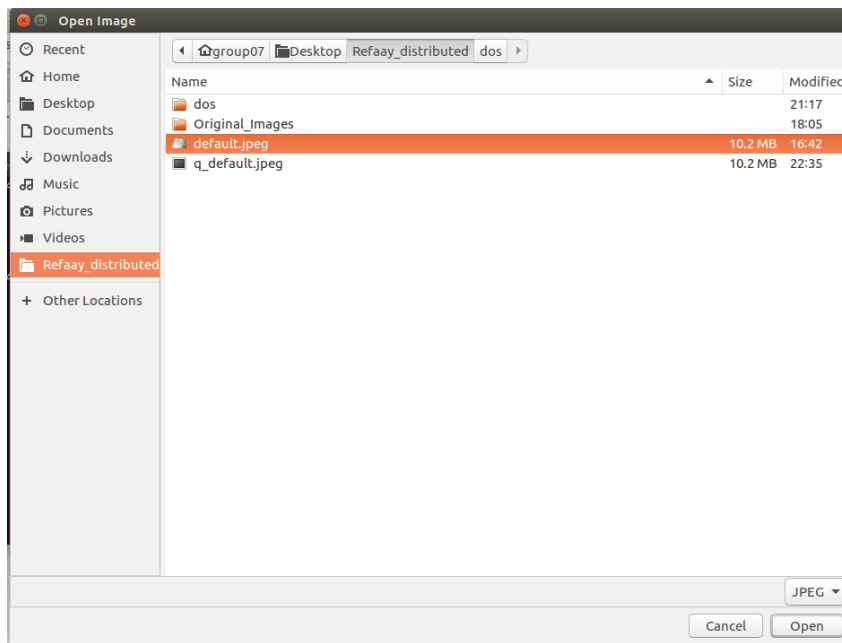
When pressing on all users, this is what the user views. Online users are in green and the offline users are in red.



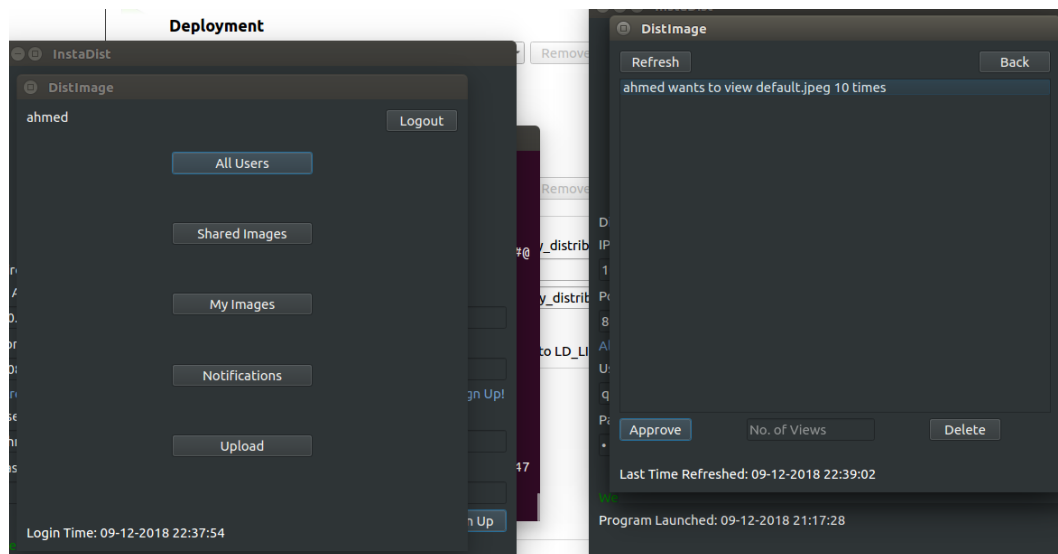
When the user selects a user this is what they view. A list of all the images they own is displayed



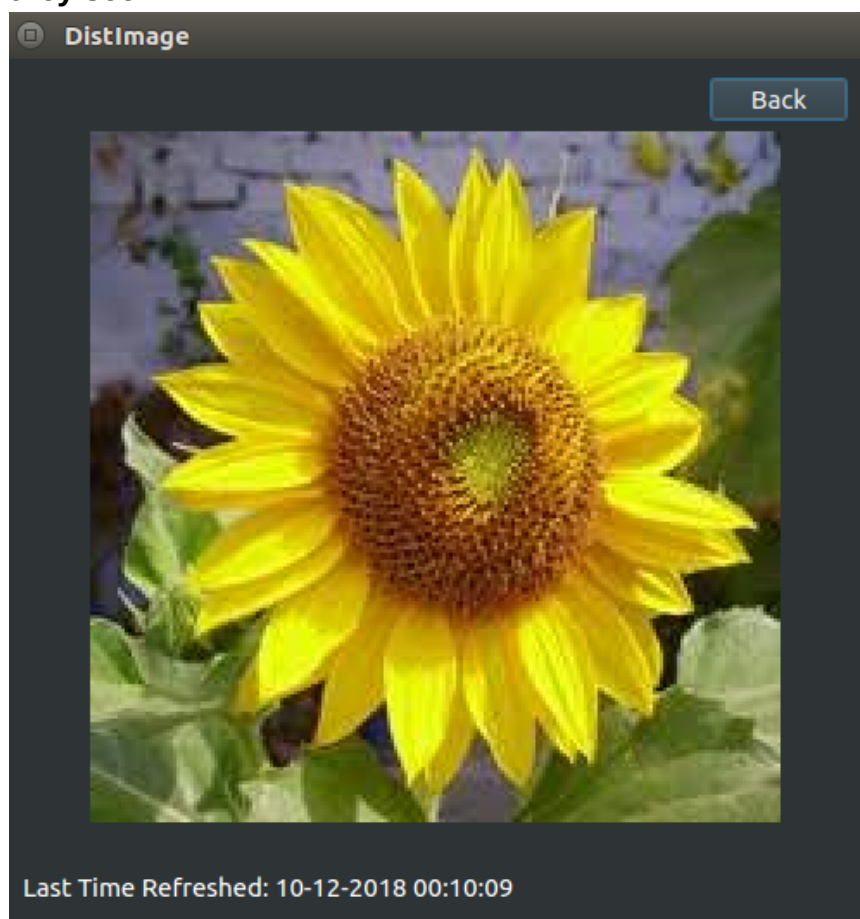
When the user chooses to upload an image, they are redirected to this browse window to find an image that you have stored locally



When a viewer requests an image from an image owner, the owner receives a notification that notifies the owner of who want to view which image and the number of times they wish to view to it.



When a viewer opens one of the images that are shared to him/her, this is what they see.



This is what an image owner would see for each image he/she owns:

Each viewer they shared the image to and the remaining number of views for each of them

