Project 3 :

Description

In this project, we created an image sharing service based on the request-reply protocol over udp messages. It enables users to register using email and password. And to upload and view their own images and the images where others users have given them permission to view.

Features

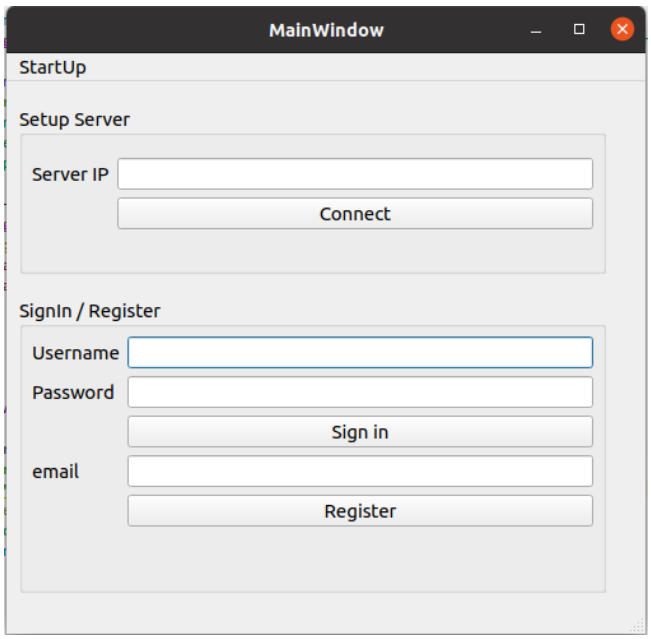
1. Registration
2. Login
3. Viewing images on local memory
4. Uploading images to the central server
5. Sending images to different users
6. Setting images counts for the number of times that a user can view a certain image.
7. That number of times gets decremented each time the user views an image.
8. Having responsive ui built using qt

Design

Front End

We have used Qt (especially Qt6) to implement the GUI of the program on the client side. The GUI contains the following windows:

a. Registration and Login Window

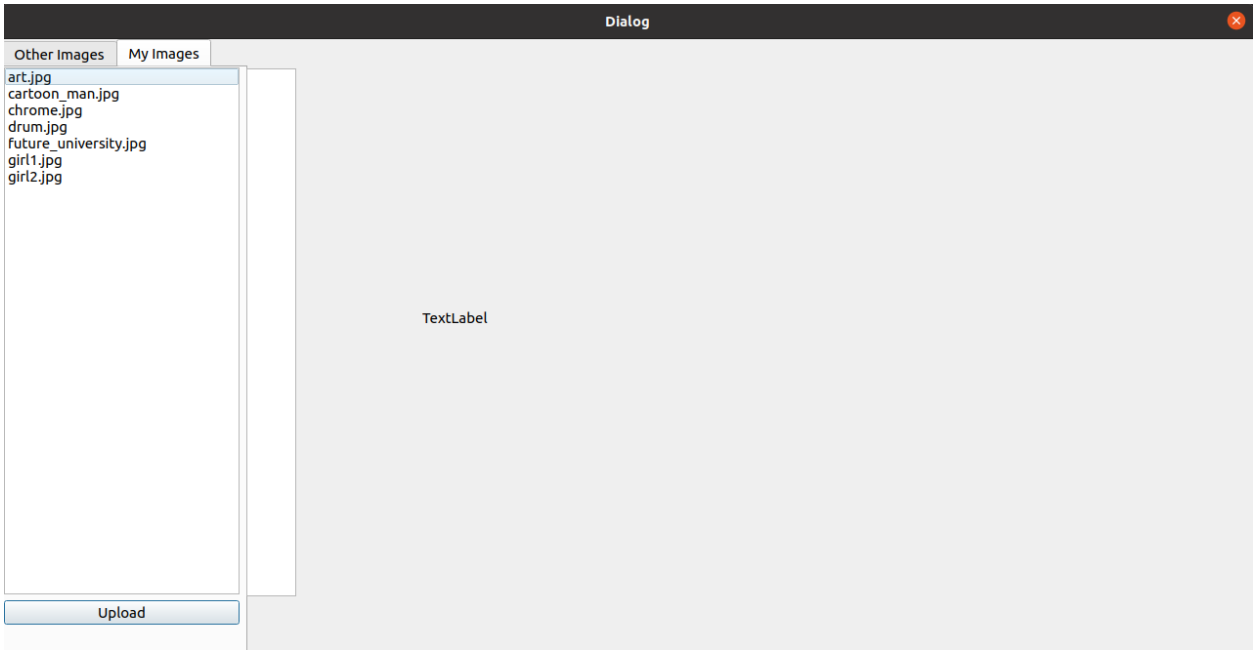


This window is responsible for three main things:

1. Connecting to the server.
   We enable a feature to enter the IP of the server, just to make sure the client connects to the server in case the server's IP has changed. Sometimes, the IP is changed due to dynamic IP protocol. The user enters the IP of the server in the server IP edit box. Then, he/she presses "connect." This connects to the server. To be clear, It doesn't show any messages. This is the first step a user must do in order to connect to the server.
2. Logging in.
   The user has to enter his username and password in the SignIn/ Register group box. After that, a request is sent to the server verifying that the user and his/her password are correct. The server checks these credentials inside the database and then sends a reply indicating whether these credentials are correct or not. In case these credentials are correct, the user is able to proceed to the main window. In case wrong, he is given another chance to change his credentials and log in again.
3. Registration:
   In this part, the user has to enter his username, password, and his email. After doing so, he has to press 'Register.' Once the user presses Register, the client automatically sends these credentials to the server to register them. If the client is already registered, he will get access to the Main window. In case, he is not registered, the server will register him inside the database.

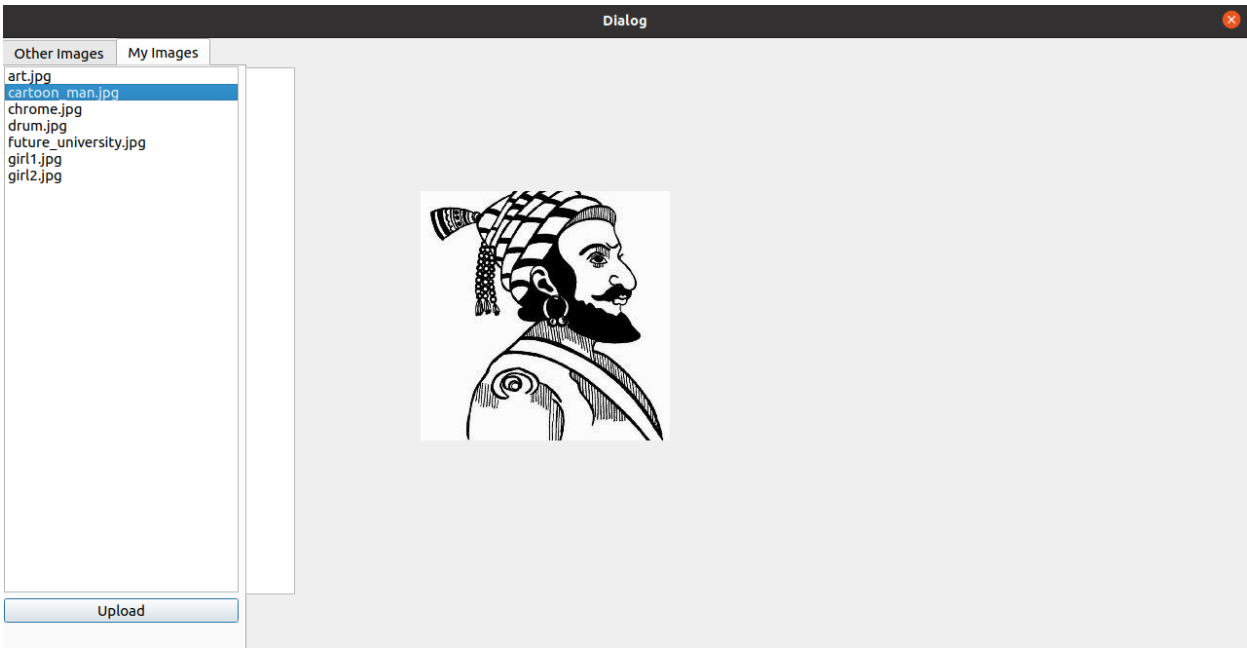
b. Main Window

After the user gets access to the application, the Main window appears, as shown below



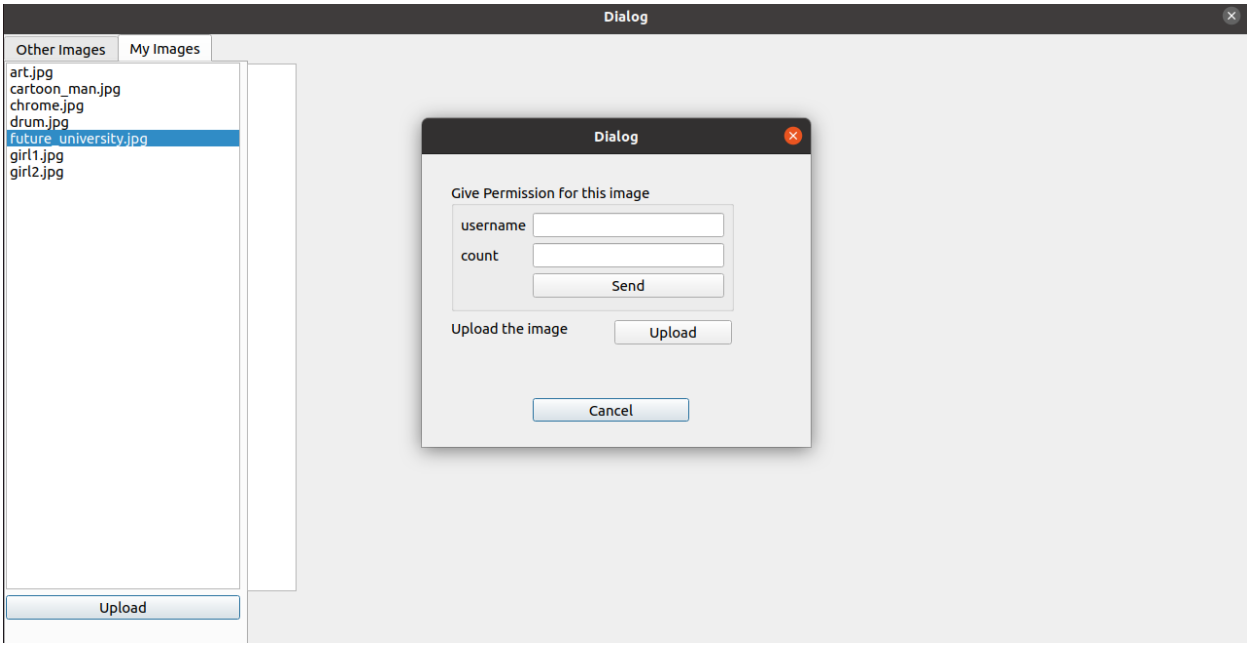
This window has four main functionalities:

1. <u>Showing the images you have currently on your device</u>
   This shows all the images that are currently inside the my_images folder. The user has to put all his images inside this directory. Once the user presses the 'My Images' tab, it shows the list of images. Once the user presses on one of them, the image is automatically displayed to the user as shown below.



2. Showing the images people are giving you access to
3. Uploading your images to the server

On the “My Images” part (by clicking on the ‘My Images’ tab), the user can upload some of these images to the server by pressing on upload button. This triggers another window, as shown below.



The user here can upload the image to the server by clicking on the upload “Upload” button. The client then sends this image to the server and registers it inside the database.

4. Give Permission to other users
   On the same dialog shown above, the user can give permission to other users in the systems. He has to type their name and the count. This information is then sent to the server to register this permission.
5. Showing the images people are giving you access to
   By clicking on the “Other Images” tab, The images the user has access to appear to the users in a list. You can see a demo for that in the demo folder especially the view_count_functionality video (https://drive.google.com/drive/folders/1jqXRYfPMt3o01t4_2SsjxScjWAcR8jUM?usp=sharing)

Back End

Our design was built on top of the implementation of the UDP client and server classes created for project 2.

In this section, I will be discussing our overall design choices along with the libraries and the tools we used.

Mainly, we created the class “ *image_server* ” which inherits from the *udp_server* class and overrides the function “server.” in the *images_server*, we defined the function *handle_requests.*

This function takes requests marshaled into a JSON format as input. Then, it checks on the field *request_type* which is an enum defined and shared between all clients and servers. If no such field exists in the JSON, we threw an exception for bad_argument.

And in case this field exists and depending on its value, we enter the corresponding section in the switch statement that servers the needed functionality.

In this project, for communication, we marshaled parameters into JSON format using the library " JSON for modern c++ ". This made sending requests very convenient.
example :

```
nlohmann::json verify;

  verify["request_type"] = RequestType::VERIFY;
  verify["email"] = "Laila@emial.com";
  verify["password"] = "123456";

  verify_user( client,  verify );
```

For the *handle_request* function, here is a snippet:

```
        int request_type = request["request_type"];

        switch (request_type)
        {
        case RequestType::ADD_USER:
            this->add_new_user_to_db(request);
            break;
        case RequestType::VERIFY:
            this->verify_user(request, client_address_);
            break;
        case RequestType::ADD_IMAGE:
            this->add_image_to_db( request  );
            break;
        case RequestType::GIVE_PERMISSION:
            this->give_permission_to_user( request  );
            Break;
```

```
. . .
```

For storing the data and monitoring the permission given to users to view images, we used an SQLite database.

We had three tables: users, images, and images_premission

```
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT ,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS images (

    image_id INTEGER PRIMARY KEY AUTOINCREMENT,
    content TEXT,
    user_id INTEGER NOT NULL,

    FOREIGN KEY(user_id) REFERENCES users(user_id)
);

CREATE TABLE IF NOT EXISTS images_permission (

    image_id INTEGER NOT NULL  ,
    sent_from INTEGER NOT NULL,
    sent_to INTEGER NOT NULL,
    view_count INTEGER NOT NULL,

    PRIMARY KEY( image_id, sent_from, sent_to  ),
    FOREIGN KEY(sent_from) REFERENCES users(user_id),
    FOREIGN KEY(sent_to) REFERENCES users(user_id),
    FOREIGN KEY(image_id) REFERENCES images(image_id)
```

```
);
```

How to run the backend server:

```
g++ images_server_example.cpp base64.cpp -o images_server_example -l pthread -l sqlite3; ./images_server_example
```

To run the frontend, you need to import the program into qt and run it from there :)

Workflow

Procedures

As the program starts, u enter the IP of the server, u can run the server locally and click connect and it will work just fine.

Then one must log in using one's own credentials.

After that, you can view to list, one with your images on your machine, and the other one with the images you are allowed to view.

—> to view this list, you must click at least once on the tab named “others images”.

Then as you click on the name of the image, a request is sent to the server, and the images is downloaded and you can view it.

At that exact moment, the number of times you are allowed to view an image is reduced by one.

And if you exhausted all the available view quota, those images get removed from the list of viewable images.

Demo

We created a view demo for our project, you can find it attached with that report (https://drive.google.com/drive/folders/1jqXRYfPMt3o01t4_2SsjxScjWAcR8jUM?usp=sharing)

Conclusion

In conclusion, we created this image-sharing service with an interactive UI using qt. Our system was designed with the right abstractions to enhance the speed of development and add new features.