Screen Share Module

Rudr Tiwari (Team Lead)

Problem Description

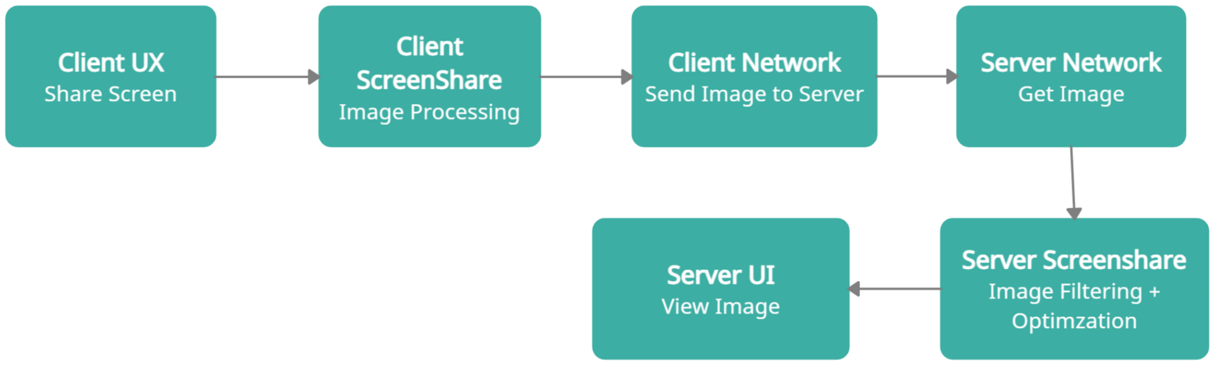
The aim is to add a screen-sharing feature to PlexShare.
Basic features needed -

1. Admin's machine should be able to view screens of all the users in a tiled manner
2. Admin should be able to pin the screen of any particular user, i.e., see its screen in enlarged view
3. Users should be able to stop and start screen sharing (full screen always)
4. Users cannot see the screen of other users
5. User will only see alert buttons
6. Mouse capturing
7. Present the screen sharing icon on the UI when screen is being presented

The module also aims to use an optimized algorithm for screen sharing which consumes the minimum possible bandwidth.

Design & Workflow

In the current implementation, the student's machines will act as client machines, whereas the administrator's machine will be the server machine. There will be only one server machine. Multiple clients may try to share their screen at the same time.



The above activity diagram shows the basic flow of how the Screenshare component is going to work. The detailed workflow and activity diagram can be found in the team members' spec. Currently, screen share is only supported by the clients and not by the server. Hence the entire flow is single directional. The challenges which will be faced in this design are mentioned below.

Proposed Implementation

To implement a screen share, the client machine needs to take screenshots of their screen at particular intervals of time. Since we want to see a live video, it should be made sure that the pace at which the screenshots are being taken is faster than what a human eye can perceive. According to research, this can be achieved by showing pictures at 24fps, and hence the target fps which we want to achieve is the same.

Challenges

Client side challenges

1. Since there are multiple clients trying to share screen at the same time, we would want to use minimum possible bandwidth
   Solution - The client will not send the entire image every time. Instead, only the changed pixels will be sent to the server.
2. We would have to send processed images based on server request (discussed below). To achieve higher frames per resolution, the processing delay between two frames should be minimal
   Solution - The images will be compressed according to the required resolution before they are sent to the server

Server side challenges

1. Since the client will only send changes every time, server needs to implement an image patching logic which changes the older frame to a newer frame
   Solution - Implement image stitching logic on server side
2. Since multiple clients are sharing their screens at the same time, we will display a maximum of nine screen tiles at a time. Based on the number of displayed tiles on the screen, the server has to request different resolutions frames from the particular client
   Solution - The viewmodel will send the number of tiles currently on display, which in turn will compute the required resolution and broadcast it to all the clients

Tasks

Client

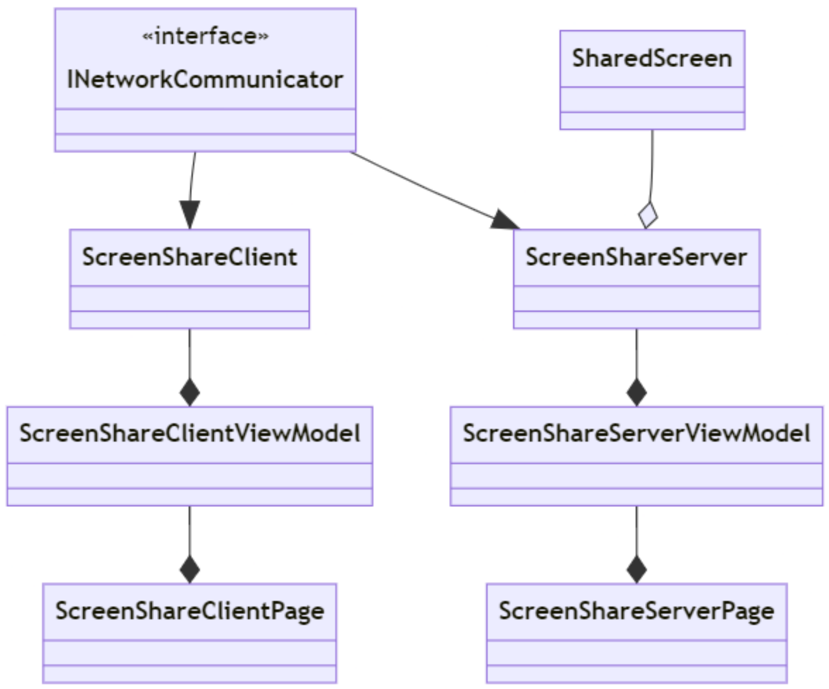
1. Client View - To display the start sharing and stop sharing buttons
2. Client viewmodel - Interaction logic between the UI and the data model
3. Image capturing - Starts a thread to capture screen at fixed interval of time
4. Image processing - Process the screens which are being captured
5. Image piping - Sends the processed screen to the Network module

Server

1. Image piping - Receive images sent from various clients over the network
2. Image processing - Use image patching to modify the current screen
3. Viewmodel - Logic to request new page of users from UI, pin users, sending the current from to UI for display
4. View - Buttons for pinning a user, shifting to next page of users, displaying the client screens into the tiles present on current screen

Interfaces and Classes

Class Diagram



The class diagram for the entire module is shown above. There will be separate view, viewmodel and model classes for each of the client and server, which will be inherited from the INetworkCommunicator interface.This interface will be provided by the networking module. Both the client and the server have to implement the OnDataReceived functions for these modules.

Classes

Below are all the methods and members for each of the classes and a brief description of what they are expected to do

Client’s View Class

```
public class ScreenShareClientViewModel {
     // Start screen share button
     StartButton_Click()

     // Stop screen share button
     StopButton_Click()
}
```

Client’s View Model Class

```
public class ScreenShareClientViewModel {
     // The constructor of this class will create an object of the
     screen share class

    // Will trigger the function `StartSharing` in the model class
    // It will be called when the client clicks on the start sharing
    // button
    public void StartSharing() {...}

    // Will trigger the function `StopSharing` in the model
    // It will be called when the client clicks on the stop sharing
    // button
    public void StopSharing() {...}
}
```

Client’s Screen Share Class

```
public class ScreenShareClient : INetworkCommunicator {

    // The Networking object to send packets and subscribe to it
    private INetworking communicator;

    // Tells whether the client is sending the packets or not
    private bool req_screen;

    // The old resolution that the client was sending
    // and the new resolution which the server is requesting
    private Resolution old_res, new_res;
    // The queue in which the image will be enqueued after
    // capturing it and processing it
    private Queue<Frame> captured_frame, processed_frame;

    // The threads that were started to capture and process the
screen
    // and to send the packets to the network
    private Thread capture_thread,
                    process_thread,
                    sendImage_thread,
                    sendConfirmation_thread;

    // The constructor for this class. It will instantiate the
    // networking object and subscribe to it for `OnDataReceive()`
    public ScreenShareClient() {...}

    // This method will be called by the ViewModel
    // Sends subscribe requests to the server
    // Call the methods `sendImagePacket` and `capnProc()`
    public void StartSharing() {...}

    // This method will be called by the ViewModel
    // Sends unsubscribe request to the server
    // Stop the currently running threads and clears the queue
    public void StopSharing() {...}

    // This method will be invoked by the networking team
    // This will be the request/response packets from the server
    public void OnDataReceive(DataObject) {...}

    // This method will start the thread in which it will read
    // the processed frames from that queue and start sending
    // those images to the server. It will only send the images
    // if the server has requested for sending the packets
    // i.e. the req_screen variable is true
    public void SendImagePacket() {...}

    // This method will start the thread which it will keep on
    // sending the packet having the header as confirmation to let
    // the server and the dashboard know that the client is not
    // disconnected and is still presenting his screen.
    public void SendConfirmationPacket() {...}

    // the main processing function
    public void CapNProc()
    // runs on the capture_thread and pushes to captured_frame
    public void Capture()

    // runs on process_thread and pushes to processed_frame
    public void Process()


     // run the compression algorithm and returns list of changes in
     pixels
     public void Compress()
}
```

Server's View Class

```
public class ScreenShareServerViewModel {
     // Handler for page change button
     public void PageChangeButton_Click()

     // Handler for button used for pinning a user
     public void PinButton_Click()
}
```

Server's View Model Class

```
public class ScreenShareServerViewModel {
    // Keeps track of the current page that the server is viewing
    public int currentPage = 1;

    // The maximum number of tiles of the shared screens
    // on a single page that will be shown to the server
    public const int MAX_TILES = 9;

    // For each client in the currently active window, it will start
a
    // separate thread in which the final processed images of clients
    // will be dequeued and sent to the view
    private List<IP, Thread> currWinClients;

    // The constructor will instantiate the `ScreenShareServer`
object
    // It will start a thread for the method `ReadImages()`
    public ScreenShareServerViewModel() {...}
    // It will compute the current window clients from the active
list
    // of the subscribers using the pagination logic, and start the
    // thread for these clients using `SendImageToView()` method
    public void ReadImages() {...}

    // It will read the final processed image of the client and send
    // them to the view. This method will run for an infinite while
    // loop and as soon as it receives an image in the queue, it
    // will dequeue it and send it to the view
    public void SendImageToView() {...}

    // Update the current page number and call the `Broadcast` method
    // in the server class. Ask the previous clients to stop sending
    // packets and stop their threads. Ask the new clients to start
    // sending their packets
    public void OnPageChange(PageNum) {...}

    // Mark the client as pinned. Switch to the page of that client
    // and call the `OnPageChange(newPage)` method
    public void OnPin(IP) {...}
}
```

Server’s Screen Share Class

```
public class ScreenShareServer : INetworkCommunicator {
    // The Networking object to send packets and subscribe to it
    private INetworking communicator;

    // The map between each IP and the shared screen object for
    // all the active subscribers
    public Map<IP, SharedScreen> subscribers;

    // The constructor for this class. It will instantiate the
    // networking object and subscribe to it for `OnDataReceive()`
    public ScreenShareServer() {...}

    // This method will be invoked by the networking team
    // This will be the response packets from the clients
    // Based on the header in the packet received, do further
    // processing as follows:
    /*
        SUBSCRIBE    --> SubscribeClient(ip)
        UNSUBSCRIBE  --> UnsubscribeClient(ip)
        IMAGE        --> Stitch(ip, converted_image)
        CONFIRMATION --> UpdateTimer(ip)
    */
    public void OnDataReceive(DataObject) {...}

    // Add this client to the map
    public void SubscribeClient(IP) {...}

    // Remove this client from the map
    public void UnsubscribeClient(IP) {...}

    // Tell the clients the information about the resolution
    // of the image to be sent and whether to send the image or not
    public void BroadcastClients(CurrWinClients) {...}

    // Function to stitch the new image over the old image
    Stitch(ip, new_res, list of : {x, y, (R,G,B)})

    // Start/Reset the timer for the client with the `OnTimeOut()`
    public void UpdateTimer(IP) {...}

    // Callback method for the timer. It will unsubscribe the client.
    public void OnTimeOut(IP) {...}
}
```

Task Division

Rudr

Helps out Mayank with ViewModel and Image Piping. Help out in unit testing. Code review and other team lead responsibilities to be handled

Mayank

Handles ViewModel and Image Piping. Image Piping will include handling requests from the server and sending the frames to the server. Will implement functions of the viewmodel class and will implement the piping functions from the model class (ScreenShareServer and ScreenShareClient)

Harsh

Handles the entire UI component of the team which includes designing the view. Will implement functions of the view class (ScreenShareClientPage and ScreenShareServerPage)

Aditya

Handles Image Processing on the server side and image compression on client side. This includes logic for image stitching on the server side. Will implement the processing related functions in ScreenShareServer and ScreenShareClient class.

Amish & Satyam

Handles Image Processing and capturing on the client side which includes improving performance of image comparison algorithms and sending only the required part of image to the server. This also includes sending the required resolution image to the server when requested. Will implement the processing related functions in ScreenShareServer and ScreenShareClient class.

Summary and Conclusions

The entire module is divided into three components for each server and client. The components are. We are following the MVVM model for the entire module.

1. View
2. ViewModel
3. Image Processing and Piping (Model)

The screen share module will be designed in such a way that it only communicates with the Networking module. All of these communications will be mostly done using the Publisher Subscriber design pattern on the Networking module.
Detailed analysis of performance and security can be found in the individual design doc of each of the team members.

References

111901030_Spec_Doc
Spec_Doc_111901008
111901005_spec_doc
ScreenShare_satyam_mishra
Screenshare_Harsh_Dubey

Future Work

1. Option to share a particular application
2. Sharing screens at different resolution
3. Multiple admin support
4. Admin should be able to share their screen in teaching mode
5. Better frame rate