Meet.me

Dashboard Module Specifications

CONTENT

OBJECTIVE

- Provide a session manager who will manage all the relevant details about a session both on the server as well as client-side.
- Provide a summarizer that will summarise the meeting held with appropriate filtering.
- Provide a telemetry analyzer that will analyze the data transferred for the discussions held on the server and present it in a neat and efficient manner.
- Provide a persistence interface that will handle storing of files and images that are generated by the telemetry analyzer and summarizer.
- The dashboard keeps track of users in the session and all the relevant information such as usernames, streams, session passwords, etc.
- Maintain the session details such as users in the session, session password, user and IP mappings, etc.

TEAM CONFIGURATION

Team Lead:

- **Handled By:** Siddharth Shah
- **Role:**
    - To design the abstract design for the dashboard module.
    - Handle the deadlines for the dashboard module and accordingly design time schedules for other team members as well.
    - Integrate the module with other modules and handle internal team conflicts or design choices.
    - Regression testing and E2E testing for the module.

Session Manager:

- **Handled By:** Rajeev Goyal
- **Role:**
    - Runs as controller on the server as well as on the client-side
    - Initialize all managers in other modules with appropriate dependencies
    - Act as an interface between all the submodules in the Dashboard module.
    - Maintain the details of all the users in the session
    - Set the users list for the Networking module to broadcast

Summary Logic:

- **Handled By:** Sairoop Bodepudi
- **Role:**
    - Design an algorithm to summarise the contents discussed during the discussion
    - Efficient model the summarizer to run efficiently whenever client asks for summary.

Telemetry:

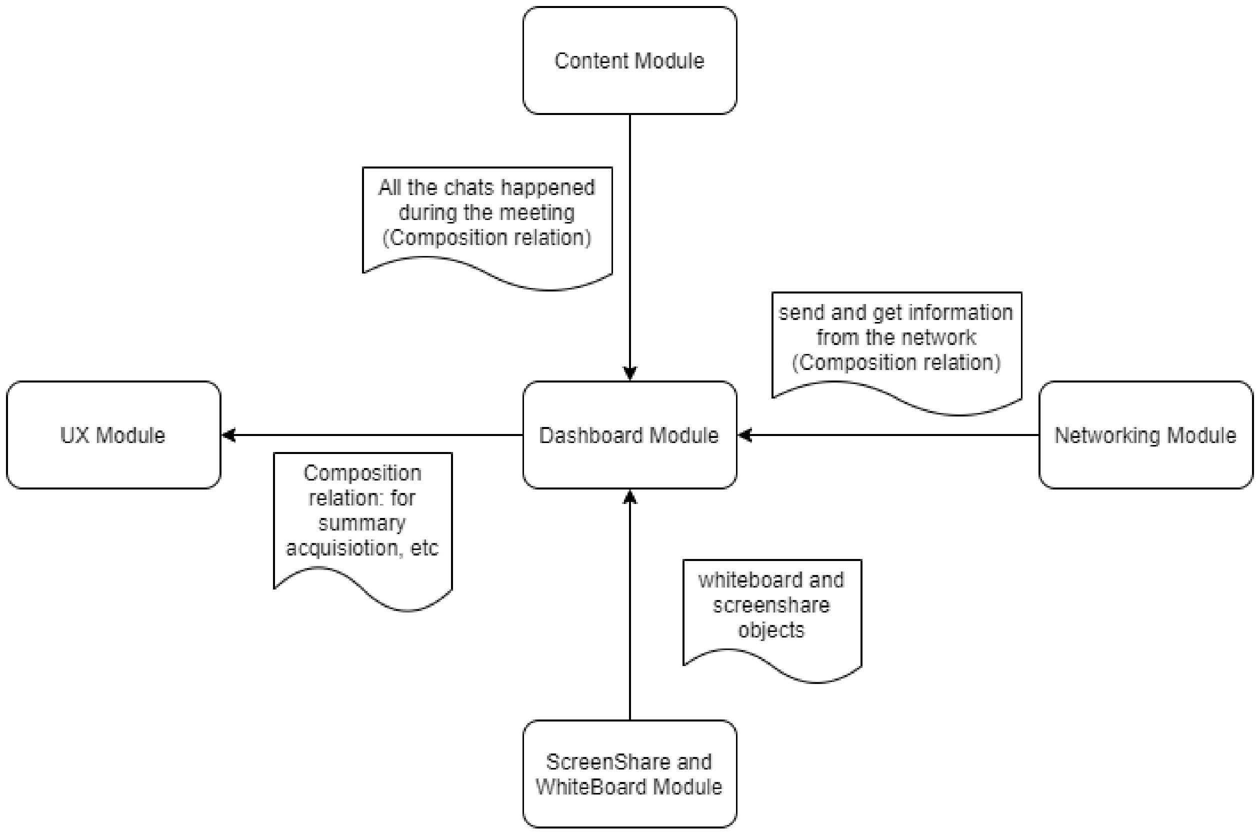
- **Handled By:** Harsh Parihar
- **Role:**
    - Run analysis on the data of all discussions done on the server
    - Run analysis on the data of a particular discussion
    - Analysis telemetrics include users/discussion, number of total discussions, chats/user for ongoing session, the time any user spends in the meeting.

Persistence:

- **Handled By:** Parmanand Kumar
- **Role:**
    - Store the summary and telemetry analysis at the end of the meeting efficiently and effectively.
    - Create graphs for the telemetry analytics and store them as in PNG format.

DEPENDENCY DIAGRAM

- The dashboard will depend on all the modules present in the project.
- Following is the dependency diagram which states the dependency relationships and the information passed during in the relation


Content Module
All the chats happened during the meeting (Composition relation)
send and get information from the network (Composition relation)
UX Module
Dashboard Module
Networking Module
Composition relation: for summary acquisiotion, etc
whiteboard and screenshare objects
ScreenShare and WhiteBoard Module

- As the dashboard module runs as a controller module, it has to initialize almost all the modules in the software

MODULE DIAGRAM

The dashboard modules have different flow tracks based on different events, all the events and the data for the event are depicted in the following diagrams.

Server Boots Up:

When the server is started the session object is initialized on the server-side by the session manager and the session manager also regulates the flow of IP address and port to connect to the server.

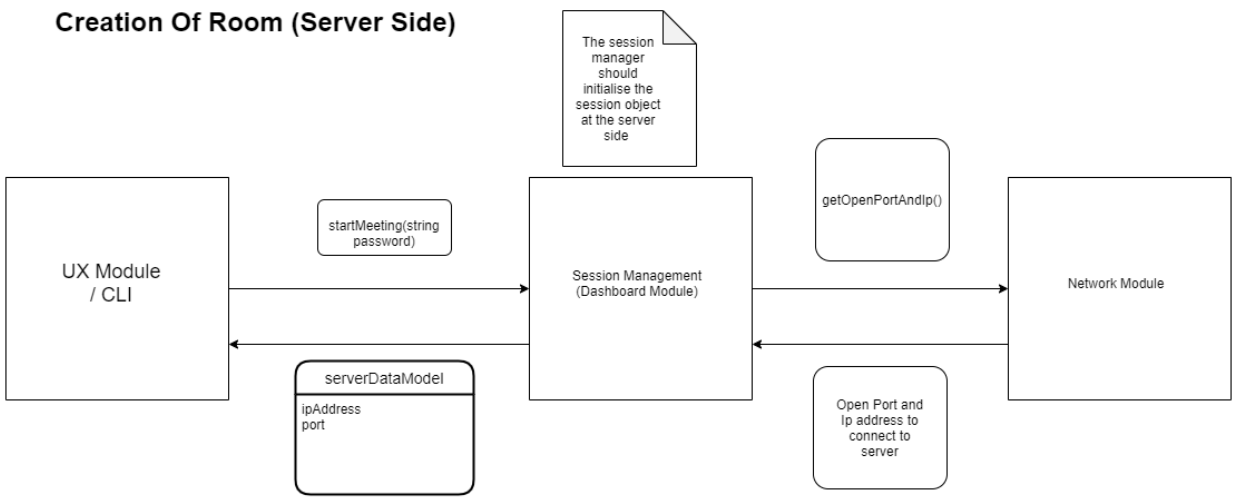

Creation Of Room (Server Side)
The session manager should initialise the session object at the server side
UX Module / CLI
startMeeting(string password)
Session Management (Dashboard Module)
getOpenPortAndIp()
Network Module
serverDataModel
ipAddress
port
Open Port and Ip address to connect to server

Figure 1. The data flow diagram whenever server boots up

Client Joins the Meeting:

When a client joins the meeting, the dashboard on the client-side receives the clientJoinModel from the UX which is transmitted to the Network Module on the client in order to send it to the server. The server-side dashboard module creates a user object based on the client details that are received and then adds this object to the session object maintained on the server-side. The telemetry submodule in the dashboard module must run its analytics on the changed data. This new user object is broadcasted to all the users who were present before the arrival of the new user and the server session object is sent to the new user. The received data at the client-side will be notified to the UX module from the dashboard module at the client-side.

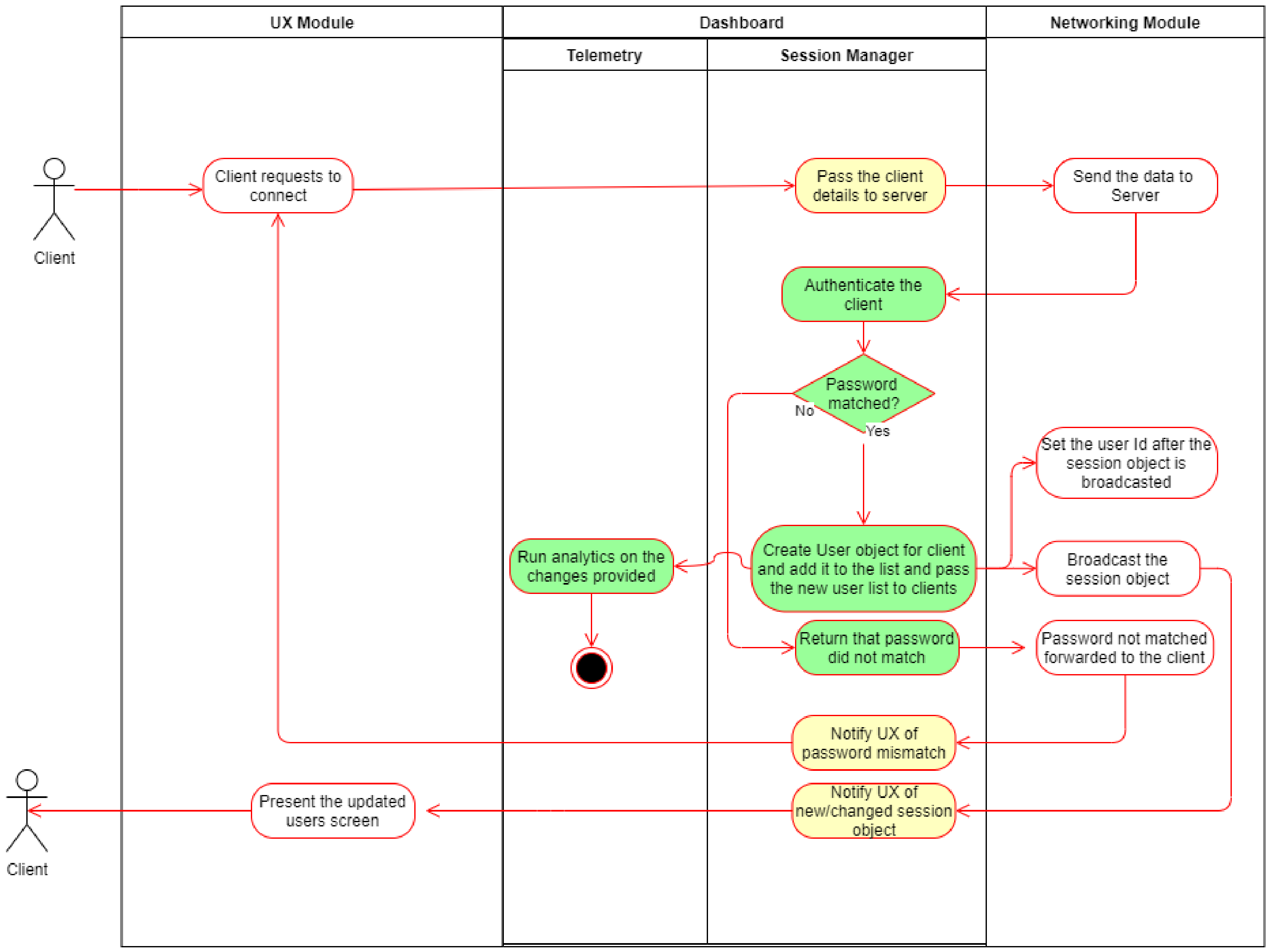

UX Module
Dashboard
Networking Module
Telemetry
Session Manager
Client
Client requests to connect
Pass the client details to server
Send the data to Server
Authenticate the client
Password matched?
No
Yes
Set the user Id after the session object is broadcasted
Run analytics on the changes provided
Create User object for client and add it to the list and pass the new user list to clients
Broadcast the session object
Return that password did not match
Password not matched forwarded to the client
Notify UX of password mismatch
Present the updated users screen
Notify UX of new/changed session object
Client

Figure 2: Activity diagram for client join event

Client Departure:

Whenever the client leaves the meeting, the UX will run the clientLeft() function from the session manager's interface for UX. This event will be passed on to the server via networking and then on the server side the dashboard module will remove the user from the session object maintained at the server-side and the same will be notified to the telemetry submodule via publisher-subscriber relationship. Then this new session object will be broadcasted to remaining clients and thus notified to the UX on other clients for the appropriate changes.

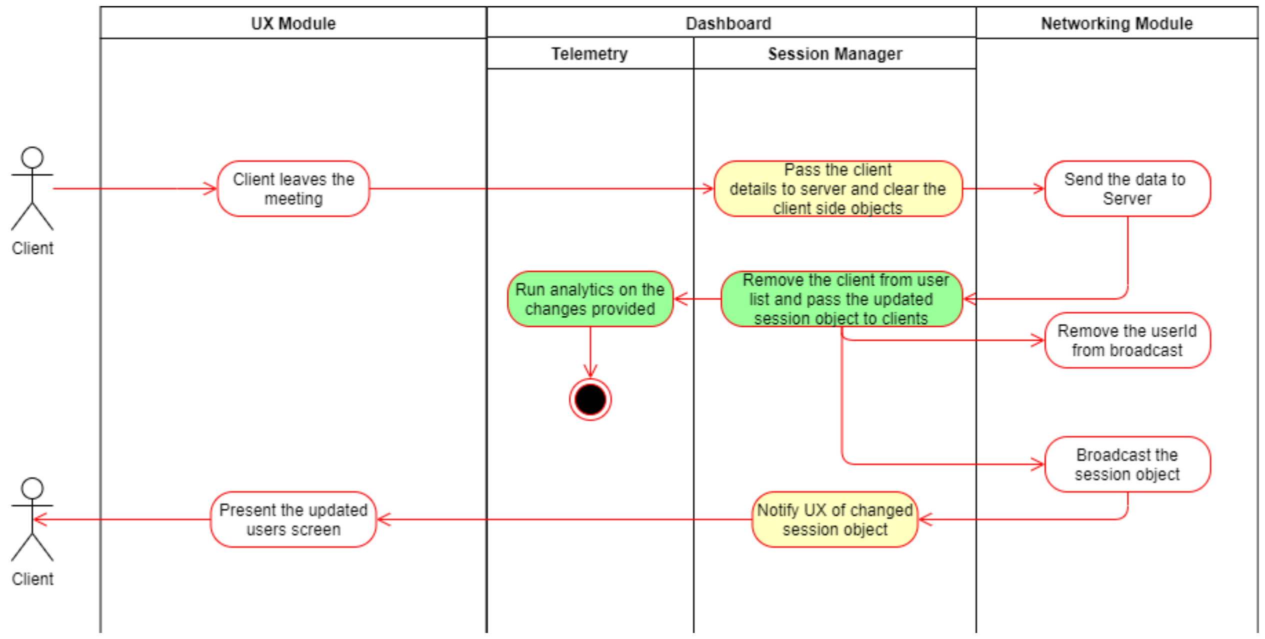

UX Module
Dashboard
Networking Module
Telemetry
Session Manager
Client
Client leaves the meeting
Pass the client details to server and clear the client side objects
Send the data to Server
Run analytics on the changes provided
Remove the client from user list and pass the updated session object to clients
Remove the userId from broadcast
Broadcast the session object
Client
Present the updated users screen
Notify UX of changed session object

Figure 3: The activity diagram for the event when some client departs

End Meeting:

When the meeting ends, the session manager should clear up the session object at the client-side. The session manager at the server-side should gather all the chats from the Chat module and provide them to the summary logic submodule and the telemetry for a summary of the meeting and analyzing the data transferred during the meeting. This summary and analysis should be given to persistence in order to store as text and png files on the host/server.

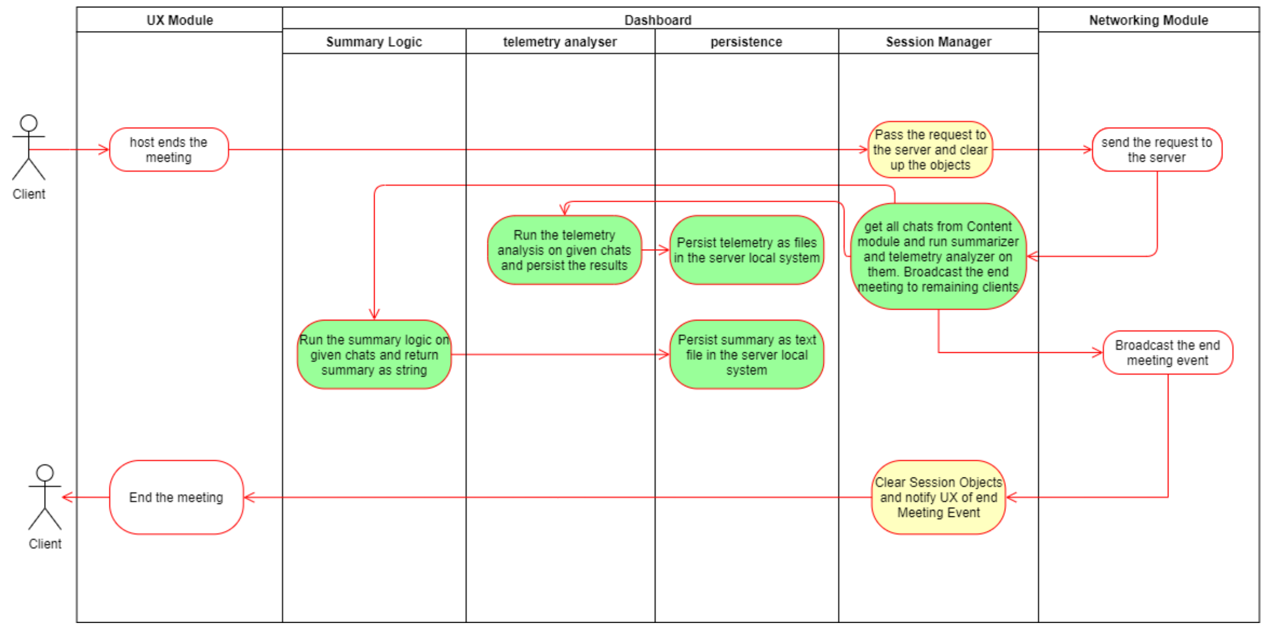

UX Module
Dashboard
Networking Module
Summary Logic
telemetry analyser
persistence
Session Manager
Client
host ends the meeting
Pass the request to the server and clear up the objects
send the request to the server
Run the telemetry analysis on given chats and persist the results
Persist telemetry as files in the server local system
get all chats from Content module and run summarizer and telemetry analyzer on them. Broadcast the end meeting to remaining clients
Run the summary logic on given chats and return summary as string
Persist summary as text file in the server local system
Broadcast the end meeting event
Client
End the meeting
Clear Session Objects and notify UX of end Meeting Event

Figure 4: a) Activity Diagram for End Meeting Event.

Summary And Telemetry Analytics Retrieval During The Session:

When some client requests the summary for the ongoing discussion, the dashboard will send the request from the client-side to the server-side, where it will run the summary logic on all the chats in discussion and return the generated summary to the client who requested it.

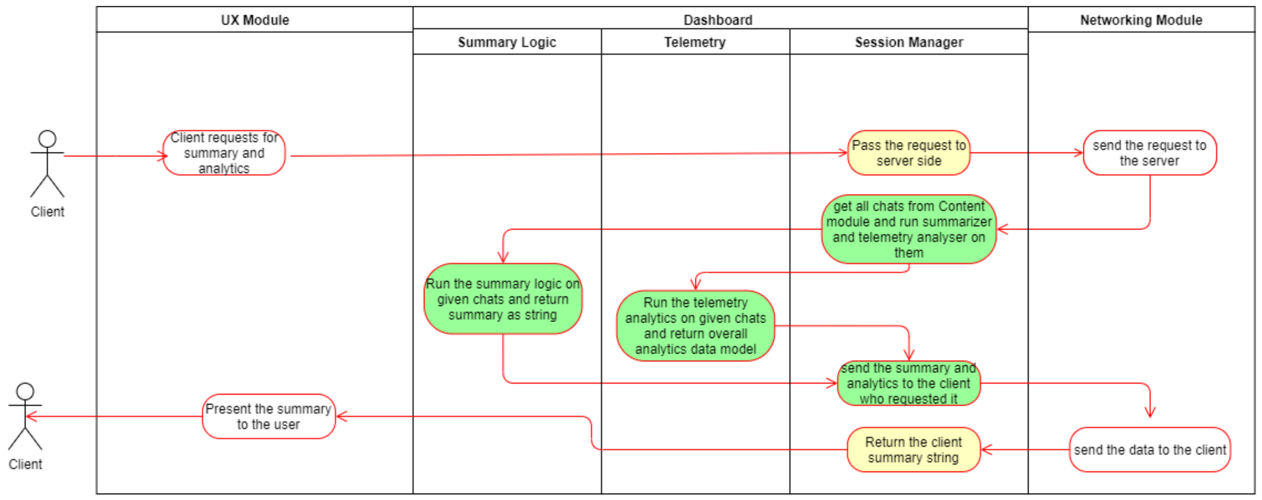

UX Module
Dashboard
Networking Module
Summary Logic
Telemetry
Session Manager
Client
Client requests for summary and analytics
Pass the request to server side
send the request to the server
get all chats from Content module and run summarizer and telemetry analyser on them
Run the summary logic on given chats and return summary as string
Run the telemetry analytics on given chats and return overall analytics data model
send the summary and analytics to the client who requested it
Client
Present the summary to the user
Return the client summary string
send the data to the client

Figure 5: a) Data flow and b) Activity Flow diagrams when some client requests for summary

<u>Connection Error:</u>

When the server is unable to send a client a message, the networking module will try to connect with the client n times and even if then the connection is not established, the client is marked as if it left the meeting. A similar procedure is applied if client is unable to connect to the server, a connection to server is tried and if not acquired then an EndMeeting Event executed at the client side

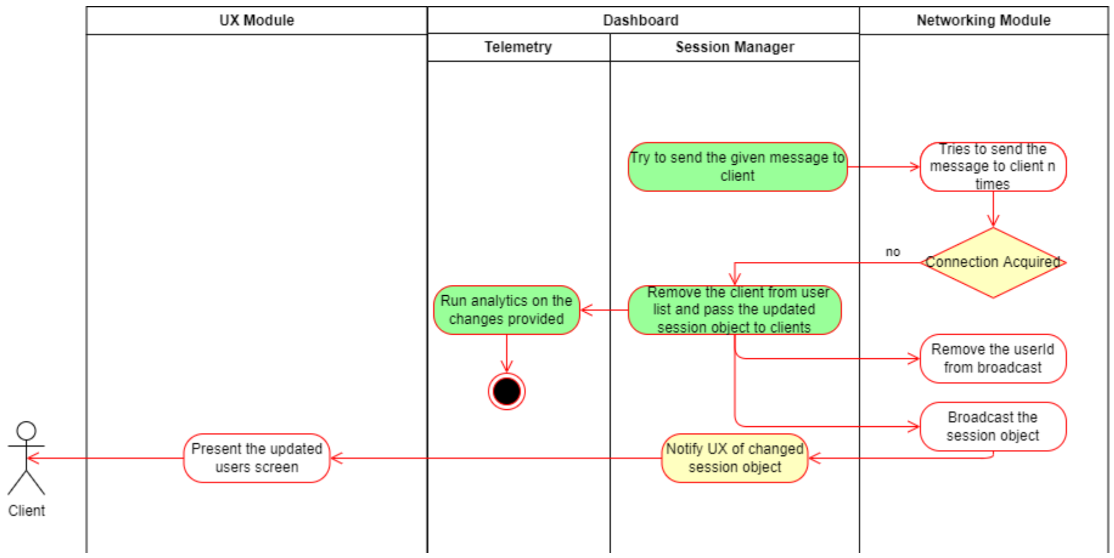

UX Module
Dashboard
Networking Module
Telemetry
Session Manager
Try to send the given message to client
Tries to send the message to client n times
no
Connection Acquired
Run analytics on the changes provided
Remove the client from user list and pass the updated session object to clients
Remove the userId from broadcast
Broadcast the session object
Present the updated users screen
Notify UX of changed session object
Client

Figure 6: a) Activity Flow diagram when server is unable to connect to some client

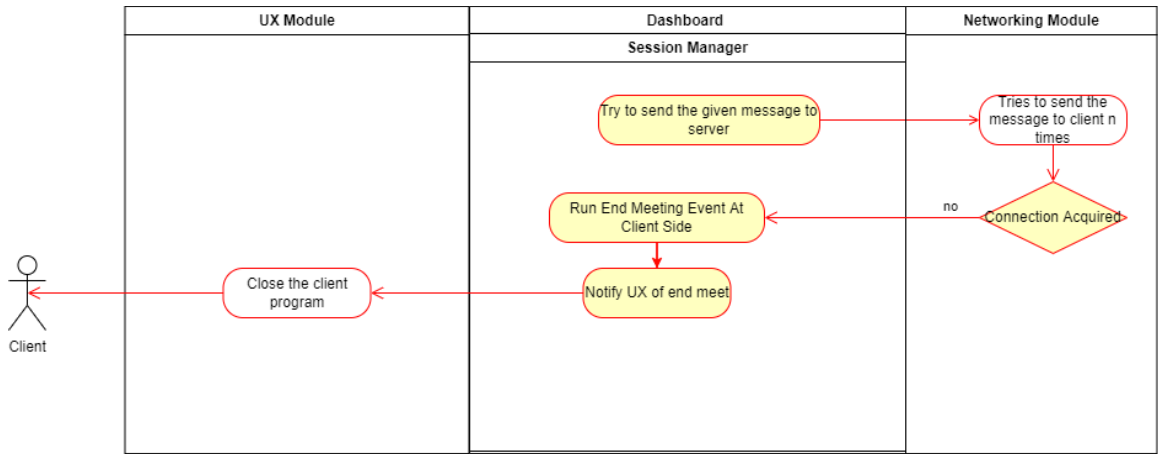


Figure 6: b) Activity Flow diagram when client is unable to connect to the server

MODULE TESTING

The module will be tested at each end point where the Dashboard module either interacts with other modules or Dashboard module stores something on the local system. To simulate how the dashboard would interact with other modules,a fake or test version of the modules which would be contacted by the Dashboard module are implemented. These fake versions would help in testing if the other modules were contacted as expected. These fake versions of other modules are provided in the constructor of SessionManager as SessionManager acts as a controller submodule for Dashboard Module and SessionManager is the only module which interacts with other modules on behalf of Dashboard Module. The parameterized constructor of session manager will only be used in the event of testing. A parameterized constructor was used because, Testing Module was using the Dashboard Module and to avoid cyclic dependency, an instance was passed in the constructors. Some features of the module had to be tested manually because of variable outputs. Following is the table which shows test cases:

Test Id	Event	Test	Expected Output	Status
1	Get IP And Port (Server Side)	Network Module returns invalid IPand port string	Return null to test UX module	Passed
2	Get IP And Port (Server Side)	Network returns valid IP and Port string	Return Meeting Credentials instance to test UX module	Passed
3	Client Arrival (Client Side)	Client provides valid IP and Port	Return true to Test UX	Passed
4	Client Arrival (Client Side)	Client provides invalid IP and Port	Return false to Test UX	Passed
5	Client Arrival (Server Side)	Network module informs dashboard module with the details of user sent from client dashboard to server dashboard	Update Network Module of new client Id and broadcast session object to all clients	Passed
6	Client Arrival (Server Side)	Network module informs dashboard module with the details of user sent from client dashboard to server dashboard	Broadcasts user object to all client where user object contains valid userId	Passed
7	Client Arrival (Client Side)	Network module informs dashboard module with the details of session sent from server dashboard to client dashboard	All the client UXes (old and new) should be notified of session object	Passed
8	Client Departure (Client Side)	When client leaves the meeting	Send an event of remove client with client details to network module to send it to server dashboard	Passed

9	Client Departure (Server Side)	Client Left event with the client details are provided to the server dashboard by Network module	Remove client id from network and session object and broadcast the session object via Network	Passed
10	End Meeting (Client Side)	When client ends the meeting	Send end meet event to network module for sending it to server dashboard	Passed
10	End Meeting (Client Side)	When Network module informs with object which indicates end meeting event	Update UX with meeting end event	Passed
11	End Meeting (Server Side)	When a end meet event is received from Network module	Saves the summary in local system	Passed
12	End Meeting (Server Side)	When a end meet event is received from Network module	Saves the telemetry analytics in local system	Passed
13	End Meeting (Server Side)	When a end meet event is received from Network module	Broadcast end meeting event to all clients via networking	Passed
14	Summary Retrieval (Client Side)	When UX wants to retrieve summary	Send an event of get summary with client details to network module to send it to server dashboard	Passed
15	Summary Retrieval (Server Side)	When a request of summary retrieval is received from Network and Chat Context is null (Chat Module Initialisation fails)	Broadcast summary as null to all clients with client details who asked for the summary	Passed
16	Summary Retrieval (Server Side)	When a request of summary retrieval is received from Network and no chats have happened until now	Broadcast summary as empty string to all clients with client details who asked for the summary	Passed
17	Summary Retrieval (Server Side)	When a request of summary retrieval is received from Network and small amount of chats have happened until now	Broadcast summary as non-empty string to all clients with client details who asked for the summary	Passed
18	Summary Retrieval (Server Side)	When a request of summary retrieval is received from Network and large amount of chats have happened until now	Broadcast summary as non-empty string to all clients with client details who asked for the summary	Passed
19	Summary Retrieval (Client Side)	When a broadcasted summary is received from network	Update UX of client who requested summary with modified summary	Passed
20	Analytics Retrieval (Client Side)	When UX wants to retrieve analytics	Send an event of get analytics with client details to network module to send it to server dashboard	Passed
21	Analytics Retrieval (Client Side)	When a broadcasted analytics is received from network	Update UX of client who requested analytics with modified analytics	Passed
22	Analytics Retrieval	When a request of analytics retrieval is received from Network	Broadcast analytics object to all clients with client details of	Passed

	(ServerSide)		the client who asked for the summary	
23	Module Initialisation (Client Side)	When the client dashboard starts	Initialise whiteboard, screen sharing and content modules	Passed

SESSION MANAGEMENT SUBMODULE

Overview

The UX layer sits at the top of the other module and users interact via this layer only. Below the UX layer is the Dashboard Module layer that handles UX's interaction with other modules. The Session Manager is an important submodule of the dashboard. It will create sessions at the beginning of the meet and manage these entities during an ongoing meeting. A server session consists of all the users currently in the meeting. The session manager is also responsible for asking the telemetry service to create/update statistics and the summariser to fetch the summary of the meeting.

Objectives

The Session manager should be able to complete the given tasks when the following events occurs:

- Host creates the meeting: The SM should give the UX the ports and IP address required to join the meeting.
- User joins the meeting: The password must be checked and the session should be updated accordingly. The content and Screen-share & whiteboard module should be provided with the new user. The Telemetry and UX module should be notified of this change.
- User leaves the meeting: The session must change accordingly. The Telemetry and UX module should be notified of this change.
- User asks for getting the summary: The SM should get the summary from the summariser by providing it with the chats of the meet. This summary then should be returned to the UX.
- Host ends the meeting for everyone: The summary and analytics are created and stored in the host's local machine.

Design Analysis

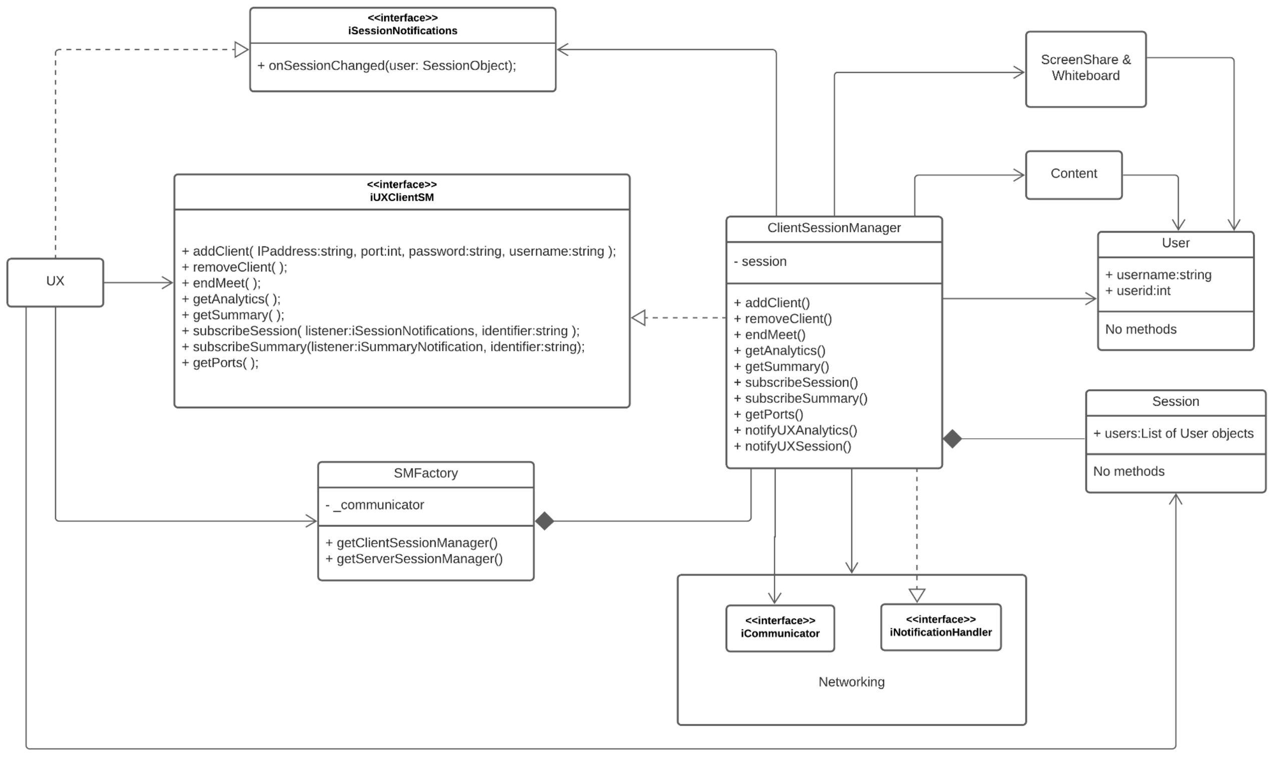
The UX depends on the Session Management module which in turn depends on the Networking Module.The SM(Session Manager) uses the Content module to get the chats and send users to it. Similarly, the Screen-share and Whiteboard module is also used by the SM to send/set users. To achieve this design in the implementation, the following can be done:

- The Session Manager(SM) will maintain a session object on the server side which will contain the list of all the users present in the meeting.
- The SM will provide the UX with a factory using which the UX can create Session manager objects (both client and the server side) that will live till the end of the meeting.
- The session management module will consist of an interface which will be used by the UX module with the help of functions defined.
- To notify the UX about any changes, the session management module will also consist of listeners for different kinds of subscriptions.
- The Networking module will provide a factory to the SM so that it can create a networking object. The networking module will provide an interface to the Session Management module that will be used by the latter.
- The Session management module will subscribe to the listener(s) provided by the Networking module.
- The Content module will also create an interface which will be used by the SM to set users or get the chat. Same goes for the Screen-share & whiteboard module.
- The Summarizer submodule and the Telemetry submodule will also provide interfaces to the SM to fetch or save the summary and analytics respectively. These objects will be created using the respective factories of these submodules. The SM will also provide the Telemetry with a listener so that any changes in the analytics are notified.

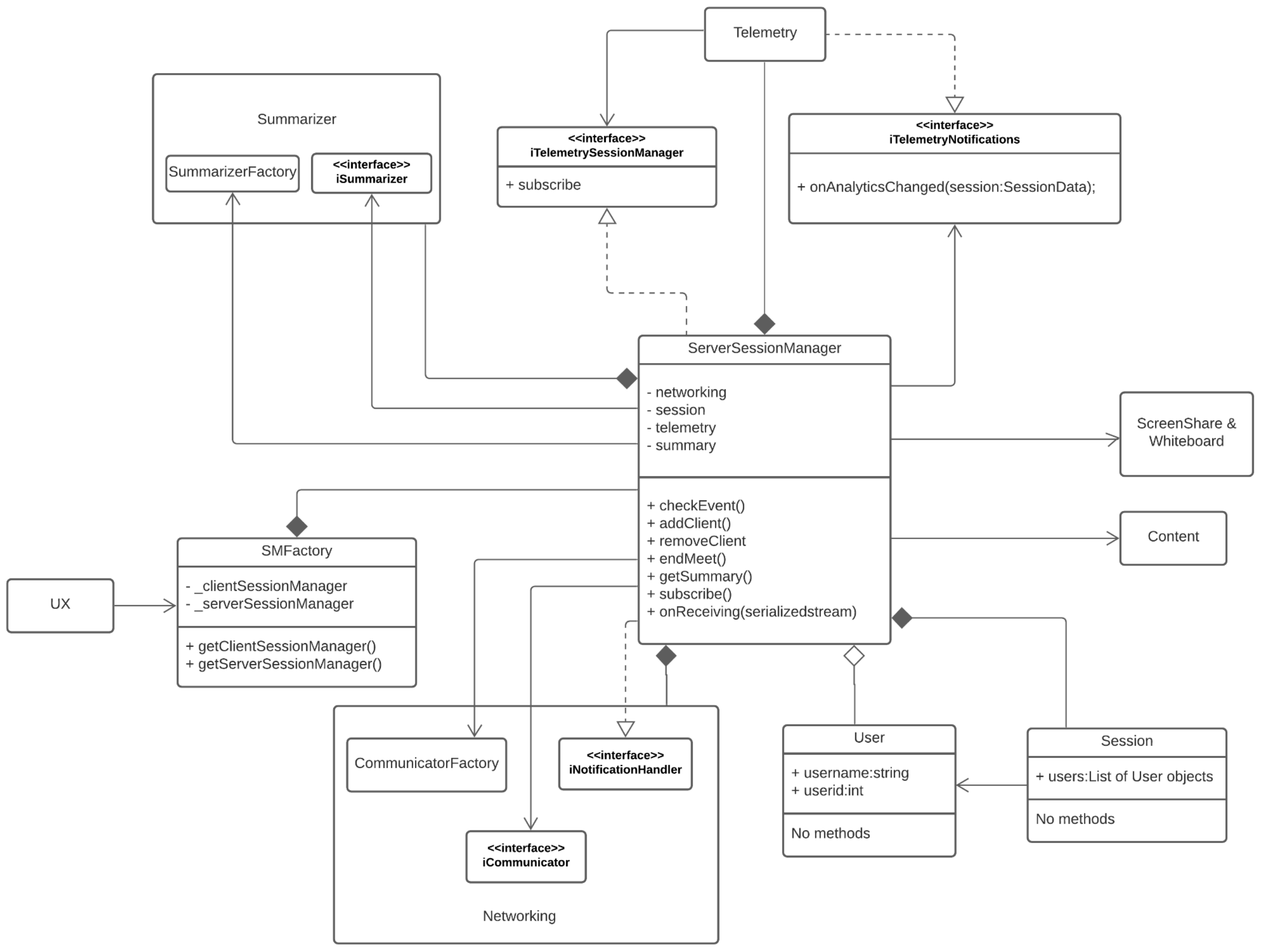
Class Diagram

For simplicity, the functions arguments are not specified in the Client Session Manager as they were already defined in the interface.

- Client Side class diagram of the Session Manager:



- Client Side class diagram of the Session Manager:



Interfaces

Client Side

```
// Notifies clients that the session has been changed
public interface iSessionNotifications
{
    // Handles the change of the session
    void onSessionChanged(SessionData users);
}

// Notifies clients that the Summary has been received
public interface iSummarynNotifications
{
    //Handles the reception of the summary
    void OnGettingSummary(string summary);
}

public interface iUXClientSM
{
    // This method will be used by the client to join
    // the meeting
    void addClient(string IPaddress,
                   int port,
                   string password,
                   string username);

    // This method is used when a client leaves
    // the meeting
    void removeClient();

    // used to end the meeting (by host)
    void endMeet();

    // Changes in the Session object can be subscribed from here
    void subscribeSession(iSessionNotifications listener,
                          string identifier);

    // This method will fetch the Summary to UX by notifying it
    void subscribeSummary(iSummarynNotifications listener,
                          string identifier);

    // It will return the ports and ipadress at the
    // beginning of the meeting so that other can join
    string getPorts();
}
```

Server Side

```
// Notifies the telemetry module about session changes
public interface iTelemetryNotifications
{
    // Handles the change of the session
    void onAnalyticsChanged(SessionData session);
}

// interface between the Telemetry and SM
public interface iTelemetrySessionManager
{
    // Subscribes to notifications from Session Management
    void subscribe(iTelemetryNotifications listener, string identifier);
}
```

Further Plans

To provide analytics on the go, using the telemetry submodule, to the UX, a class called Analytics can be made that will be returned to UX upon request. The UX can use this class to display the analytics.

SUMMARY LOGIC SUBMODULE

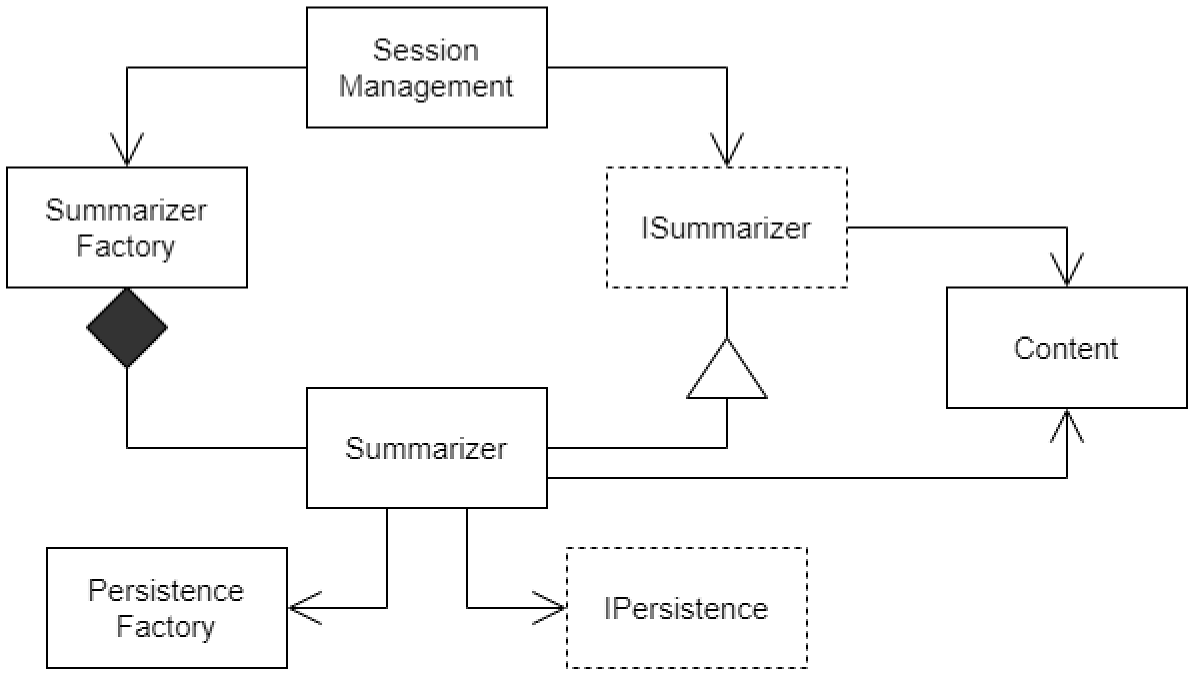
Overview

In our application, we wish to support summarizing the discussion of the session using chat to provide the user with a summary and gist of the discussion that has happened which can help in the revision of selected discussions. This module would be used by the Session management to generate summaries of different discussions.

Objective

- A single interface that can be used by the session management module:
    - This module should be simple and scalable with easy-to-use methods for session management.
    - There should be only one instance of the summarizer that carries out the predefined logic keeping track of the state and the prior while summarizing.
- The module should be running asynchronous in the server to generate a better summary:
    - The module should have methods and instances that run in the background server and not on the fly so that it would not bottleneck the performance of the application.
- The module should be able to generate summary statistics that can be influenced by the user preferences:
    - The summary generated should depend on the chat discussion in the room which would be provided after the end of the discussion and the logic can summarize it using the predefined logic.

Class Diagram



Interface

```
// Interface for Summary Logic Module

public interface ISummarizer {

    // Function to get the summary of the chat

    // and discussion to present in the Dashboard

    string GetSummary(Chat[] chats);



    // Function to save the summary of the entire

    // meeting after the completion via Persistence

    int SaveSummary(Chat[] chats);

}
```

The GetSummary function returns the string which is the summary of the chat without the presence of any metadata and sender information. This is a design choice taken since it is not required to store such information when generating the summary.

Design Analysis

Creation of Summarizer

- The creation of the summarizer is done by using the Summarizer factory which creates an instance of the summarizer which would thus persist throughout the session.
- This summarizer would be used by the Session management since the session manager would be responsible for obtaining the chat messages and other information and redirecting it to various modules.
- The interface of the summarizer is essential along with the summarizer factory since we would want the other modules to access only the different methods that would be required for summarizing and at the same time the so-called summarizer would be obtained by calling the GetSummarizer method of the summarizer factory.
- The abovementioned design choice would provide an abstraction of the internal summarizer class which would not be exposed to other modules as only its interface functions are needed and the other methods would not be required for the functionality.

Summarizer

- The Summarizer class would run the summary logic algorithm which can be data-driven or predetermined based on the requirements after the end of the discussion.
- This particular design would not bottleneck the performance of the entire application waiting for the algorithm to complete execution and it can be done at a later point in time. This also facilitates better summarization of the discussion since the algorithm would be exposed to the entire discussion/ chat while giving the summary a wider picture.

- The first method that would be available to the session manager would be the GetSummary which would be a method that would take a chat string input and provide a concise representation of the chat that would be displayed on the dashboard.
- The Summary logic algorithm would be running a probabilistic algorithm under the hood which wouldn't be requiring any dataset or prior information for the summarization of the chat.
- The generated summary would be used by the session manager to store in the database and retrieved for showing on the dashboard.
- This particular Summarizer runs only on the server side and not on the client side and this is to keep the summary processing only on the server side thereby reducing the need for processing on the client side and the client gets the summary through the UX and Session Manager.

Algorithm

The algorithm that has been designed for the chat summarizer is as follows:

- For each given chat message we find the semantic meaning and associate it with a score based on the count statistics.
- We first find all the tokens using regular expression matching and after breaking it into sentences we also find the tokens of each sentence which would be the building blocks of the sentences.
- We find the lemmas associated with each of the words using the Porter stemming algorithm, more information about which can be found here.
- Upon finding all the lemma tokens we use this to obtain a count based vocabulary which will help determine the score which after being normalized is used by the sampler to generate samples.
- Note that the algorithm that has been designed is a randomized algorithm and the summary samples are obtained based on the probability scores associated with each of the sentences which is found using the normalized scores.

Stretch Code

Multiple directions of work in the future include:

- Incorporation of more information for the summary logic algorithm in addition to the chats for example screen share, whiteboard data, etc.
- Having sophisticated data-driven Machine Learning algorithms that can be used to summarize the collected data while keeping in mind the computational barriers and making the most efficient use of the same.
- Addition of more features including recommendation systems based on the collected data to improve user experience.

TELEMETRY SUBMODULE

Overview

The function of the telemetry module is to provide analytics and statistics related to session and after the sever ends, in an easy to observe format, like histogram or pie chart. This sort of gives the overall activeness of a particular session and the load over whole server.

Objective

To fetch the information from the session management using pub-sub model, i.e. it notifies of any changes to me. Then use it to do the following analysis (for each session):

1. Plot the histogram for user vs Timestamp.
2. Plot the pieChart for user vs chat_sent.
3. Provide the list of users who were there in the session but for very less time(less than threshold time). That means, it checks that all the users who were present in the session for more than a threshold time, would only be considered as present in the meeting, else they would be returned in a list.

Forward the statistics to the persistence module for storing, and forward the statistics object to the UX module so that the statistics can be shown (it is sent via session management). Also when the server ends, I need information about all the sessions held on that server, for that I retrieve the previous session's data from the persistence. Then run the following analytics on the whole server:

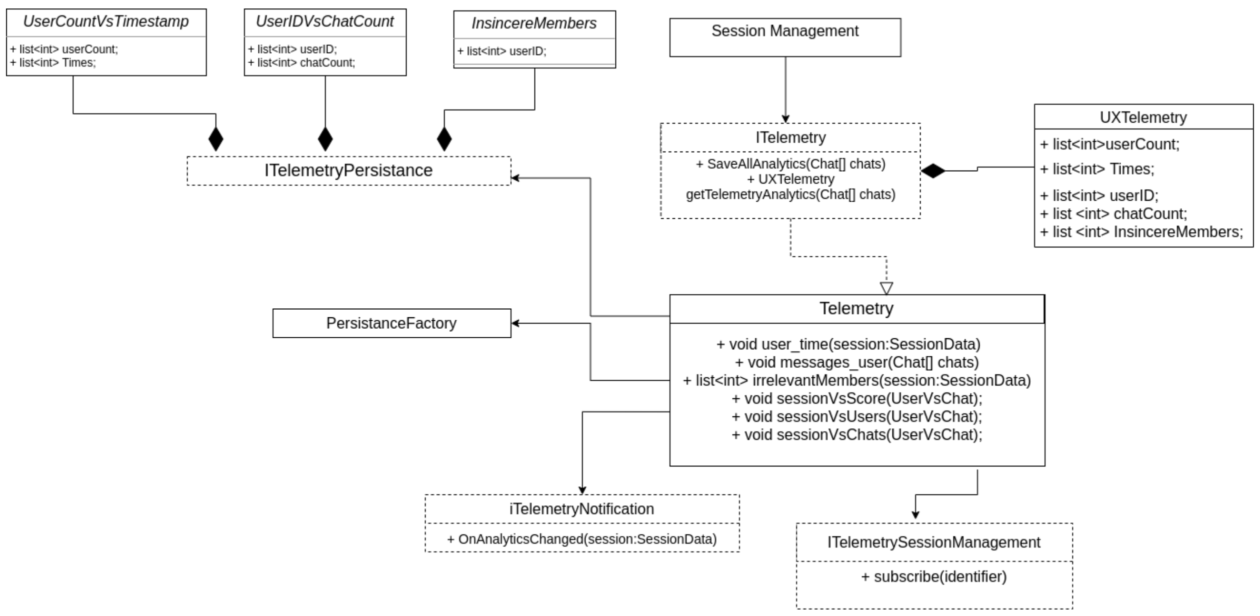
1. Session vs Score: It calculates the score of each session by some logical formula and plots the histogram for it.
2. Session vs users: Plots the histogram for number of users in each session.
3. Session vs chats: Plots the pie chart for number of chats in each session.

Then forward these graphs to the UX through the session manager.

Design Analysis

- To get the data from session management, I would be subscribing the ITelemetrySessionManagement interface, so that whenever any changes occur I would be provided with it. Hence Telemetry would be using the ITelemetrySessionManager interface.
- Telemetry class implements all the plotting functions, and for storing, I will be calling the methods of ITelemetryPersistance, hence, it will be using the ITelemetryPersistance interface. ITelemetryPersistance will get three objects of the classes “UserVsTimestamp” , “UserVsChat” and “InsincereMembers” as parameters of the function.
- Whenever the session ends, the Session Management module will call the ITelemetry’s SaveAllAnalytics() method, so that the analytics and statistics of the session can be saved with persistence(because only the session management knows when the session ends). Also, session management will call the getTelemetryAnalytics() method to take all the analytics of the session and forward it to UX module, so it will be using the ITelemetry interface.

Class Diagram



UserCountVsTimestamp
+ list<int> userCount;
+ list<int> Times;
UserIDVsChatCount
+ list<int> userID;
+ list<int> chatCount;
InsincereMembers
+ list<int> userID;
Session Management
ITelemetryPersistance
ITelemetry
+ SaveAllAnalytics(Chat[] chats)
+ UXTelemetry getTelemetryAnalytics(Chat[] chats)
UXTelemetry
+ list<int>userCount;
+ list<int> Times;
+ list<int> userID;
+ list <int> chatCount;
+ list <int> InsincereMembers;
PersistanceFactory
Telemetry
+ void user_time(session:SessionData)
+ void messages_user(Chat[] chats)
+ list<int> irrelevantMembers(session:SessionData)
+ void sessionVsScore(UserVsChat);
+ void sessionVsUsers(UserVsChat);
+ void sessionVsChats(UserVsChat);
iTelemetryNotification
+ OnAnalyticsChanged(session:SessionData)
ITelemetrySessionManagement
+ subscribe(identifier)

Interfaces

```
public interface ITelemetry{
   public void SaveAllAnalytics(Chat[] chats);
   public UXTelemetry getTelemetryAnalytics();
}
```

Further Plans

1. To store users priority wise according to messages they sent. This would be useful to find the active students in the sessions and inactive students.
2. To come up with innovative ideas of analysis which will add more value to the project, and hence can be more useful.

PERSISTENCE SUBMODULE

Overview

Persistence module is one of the important modules in the dashboard, main function of this module includes saving the Analytics, Histograms etc in the png format in each session and also summary of the chats to the local file system as.txt, when the meeting session ends. It is directly connected to other two modules namely Summary and Telemetry. These modules are supposed to call save and retrieve functions in order to persist the summaryChats and Analytics images.

Objective

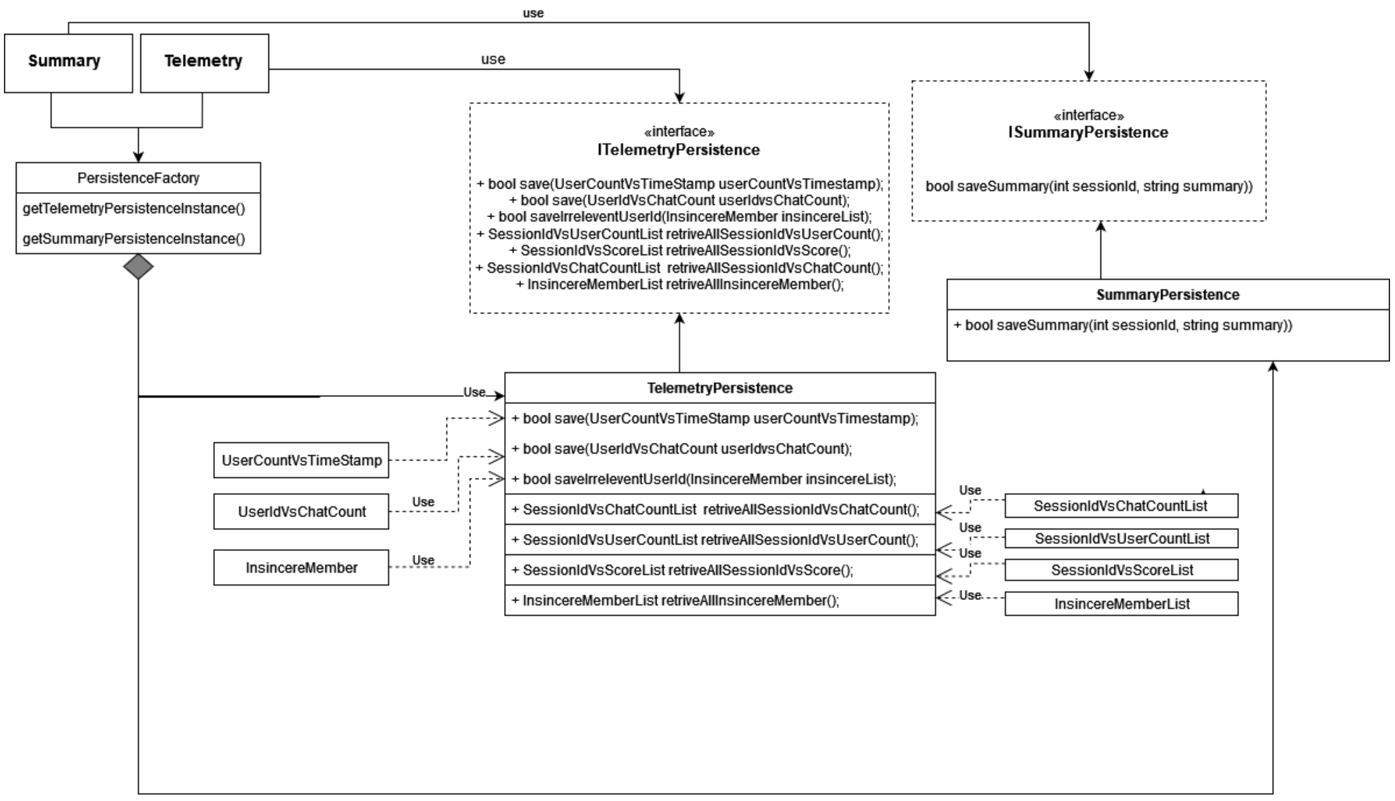
- Stores the Analytics images in the .png format session wise.
- Stores the Analytics data in a meaningful way in .txt files in each session.
- Retrieves the above Server Analytics data and sends it back to the Telemetry module at the end of the meeting.
- Stores the ChatSummary in the .txt format at the end of the meeting.

Design Analysis

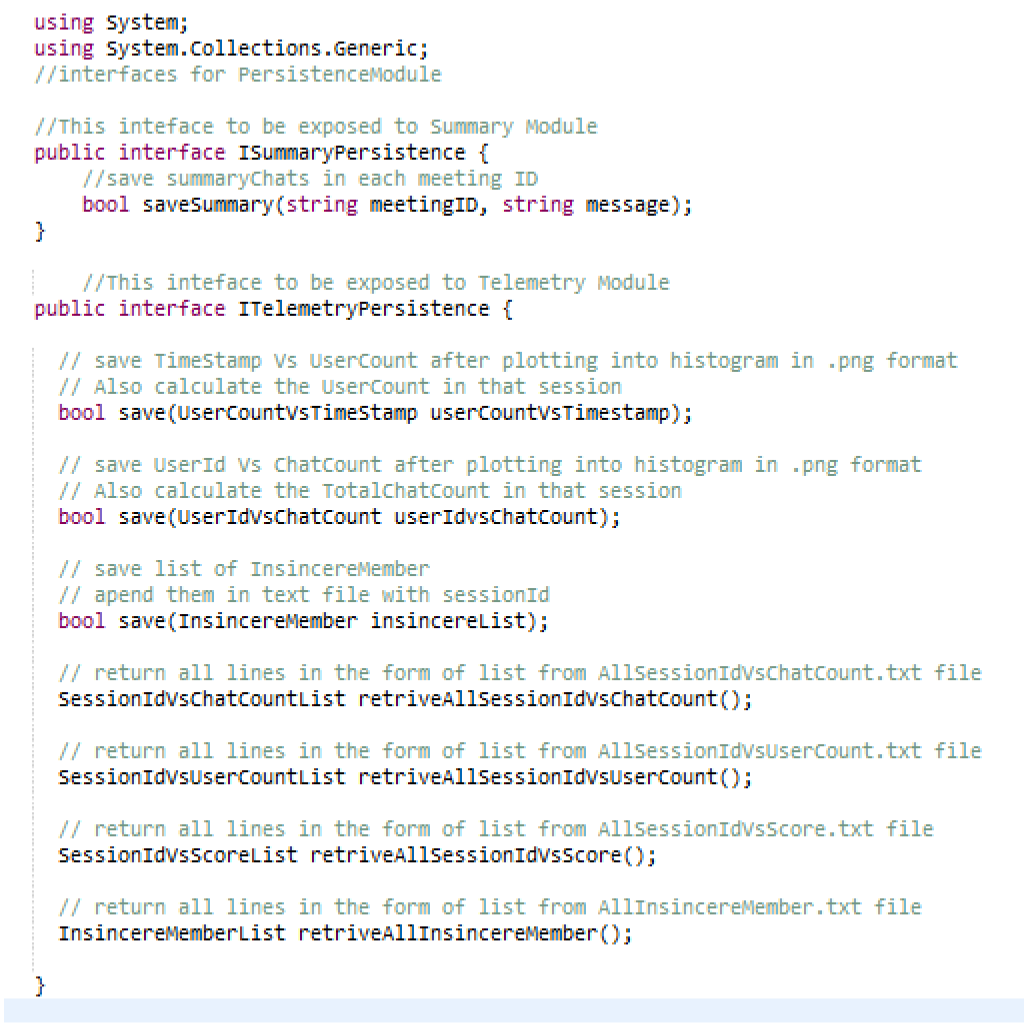
- The Persistence Module needs Histogram data as input and creates a histogram plot in order to store them as .png.
- It must also get the data in the form of string(More preferably) from Summary Module.
- This module should expose IPersistence interfaces to the other two modules and it is good to have a Factory model to create an abstraction layer while creating objects.
- There should be different save functions to store different files like in case of storing the summaryChat, we need different inputs from the other case in which it is responsible to store Histogram.
- To serve this purpose, it is better to create two different Interfaces like ISummaryPersistence and ITelemetryPersistence.
- Summary and Telemetry module should be using their corresponding interfaces.

- In case of telemetry, It is also required to store the incoming ChatCount dataVs userID data and other Analytics data in text file line by line for each session as Telemetry will require that data at the end of the meeting to create Server analytics.
- To serve the above purpose, along with saving the images, it is also necessary to store these data in text files, session wise and also At the end of meeting, it has to read all those files to return them in the form of list these analytics stored data to display server Analytics.

Class Diagram


use
Summary
Telemetry
use
PersistenceFactory
getTelemetryPersistenceInstance()
getSummaryPersistenceInstance()
«interface»
ITelemetryPersistence
+ bool save(UserCountVsTimeStamp userCountVsTimestamp);
+ bool save(UserIdVsChatCount userIdvsChatCount);
+ bool saveIrreleventUserId(InsincereMember insincereList);
+ SessionIdVsUserCountList retriveAllSessionIdVsUserCount();
+ SessionIdVsScoreList retriveAllSessionIdVsScore();
+ SessionIdVsChatCountList retriveAllSessionIdVsChatCount();
+ InsincereMemberList retriveAllInsincereMember();
«interface»
ISummaryPersistence
bool saveSummary(int sessionId, string summary))
SummaryPersistence
+ bool saveSummary(int sessionId, string summary))
Use
TelemetryPersistence
+ bool save(UserCountVsTimeStamp userCountVsTimestamp);
+ bool save(UserIdVsChatCount userIdvsChatCount);
+ bool saveIrreleventUserId(InsincereMember insincereList);
+ SessionIdVsChatCountList retriveAllSessionIdVsChatCount();
+ SessionIdVsUserCountList retriveAllSessionIdVsUserCount();
+ SessionIdVsScoreList retriveAllSessionIdVsScore();
+ InsincereMemberList retriveAllInsincereMember();
UserCountVsTimeStamp
UserIdVsChatCount
Use
InsincereMember
Use
Use
SessionIdVsChatCountList
Use
SessionIdVsUserCountList
Use
SessionIdVsScoreList
Use
InsincereMemberList

Interfaces



```
using System;
using System.Collections.Generic;
//interfaces for PersistenceModule

//This inteface to be exposed to Summary Module
public interface ISummaryPersistence {
    //save summaryChats in each meeting ID
    bool saveSummary(string meetingID, string message);
}

    //This inteface to be exposed to Telemetry Module
public interface ITelemetryPersistence {

  // save TimeStamp Vs UserCount after plotting into histogram in .png format
  // Also calculate the UserCount in that session
  bool save(UserCountVsTimeStamp userCountVsTimestamp);

  // save UserId Vs ChatCount after plotting into histogram in .png format
  // Also calculate the TotalChatCount in that session
  bool save(UserIdVsChatCount userIdvsChatCount);

  // save list of InsincereMember
  // apend them in text file with sessionId
  bool save(InsincereMember insincereList);

  // return all lines in the form of list from AllSessionIdVsChatCount.txt file
  SessionIdVsChatCountList retriveAllSessionIdVsChatCount();

  // return all lines in the form of list from AllSessionIdVsUserCount.txt file
  SessionIdVsUserCountList retriveAllSessionIdVsUserCount();

  // return all lines in the form of list from AllSessionIdVsScore.txt file
  SessionIdVsScoreList retriveAllSessionIdVsScore();

  // return all lines in the form of list from AllInsincereMember.txt file
  InsincereMemberList retriveAllInsincereMember();

}
```

Further Plans

If time permits and feasible to do it, it will also store screen share data as well.

FUTURE SCOPE

- Add screen share images and whiteboard objects to the summary of the meeting.
- Add more telemetry analytics such as latency in the network, etc in the final meeting analysis