

Meet.me

## Dashboard Module Specifications

# CONTENT

<b>OBJECTIVE</b>	<b>4</b>
<b>TEAM CONFIGURATION</b>	<b>4</b>
Team Lead:	5
Session Manager:	5
Summary Logic:	5
Telemetry:	6
Persistence:	6
<b>DEPENDENCY DIAGRAM</b>	<b>7</b>
<b>MODULE DIAGRAM</b>	<b>8</b>
Server Boots Up:	8
Client Joins the Meeting:	8
Client Departure:	9
End Meeting:	10
Summary And Telemetry Analytics Retrieval During The Session:	11
<b>SESSION MANAGEMENT SUBMODULE</b>	<b>12</b>
Overview	12
Objectives	12
Design Analysis	13
Class Diagram	14
Interfaces	16
Further Plans	17
<b>SUMMARY LOGIC SUBMODULE</b>	<b>18</b>
Overview	18
Objective	18
Class Diagram	19
Interface	19
Design Analysis	20
Creation of Summarizer	20
Summarizer	20
Stretch Code	21
References	21
<b>TELEMETRY SUBMODULE</b>	<b>22</b>
Overview	22
Objective	22
Design Analysis	23

Class Diagram	23
Interfaces	24
Further Plans	24
<b>PERSISTENCE SUBMODULE</b>	<b>25</b>
Overview	25
Objective	25
Design Analysis	25
Class Diagram	26
Interfaces	27
Further Plans	27
<b>FUTURE SCOPE</b>	<b>28</b>

## 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.

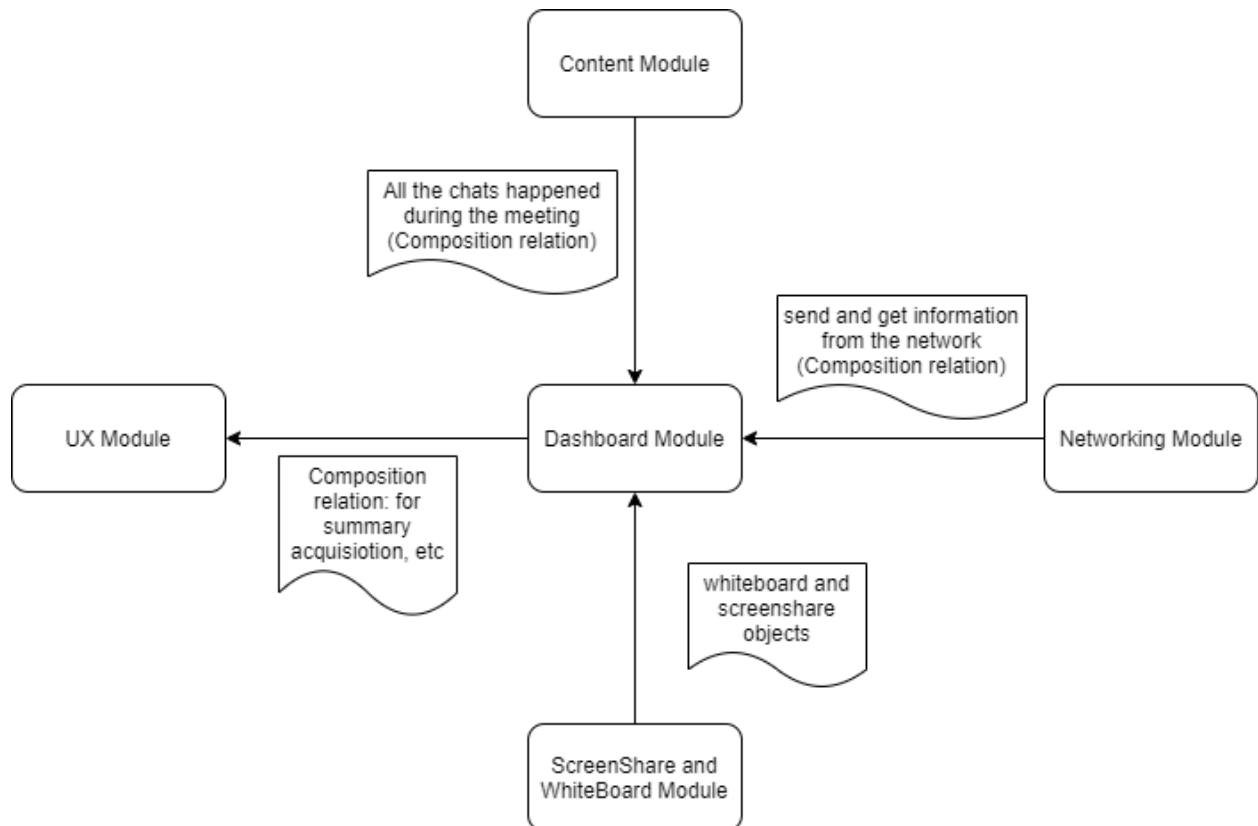
### **Persistence:**

- **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



- 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.

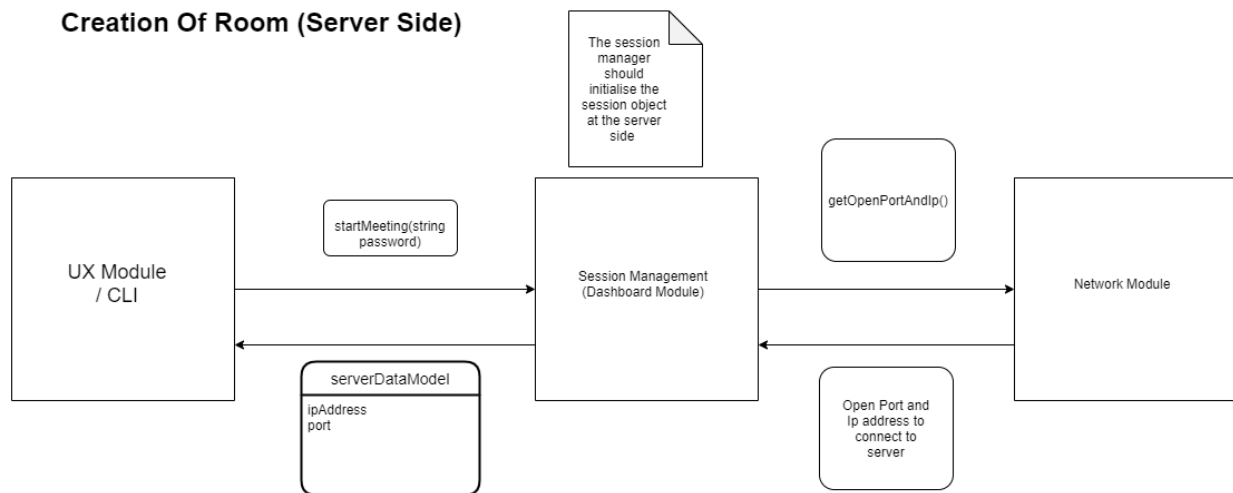


Figure 1. The data flow diagram whenever server boots up

### 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.



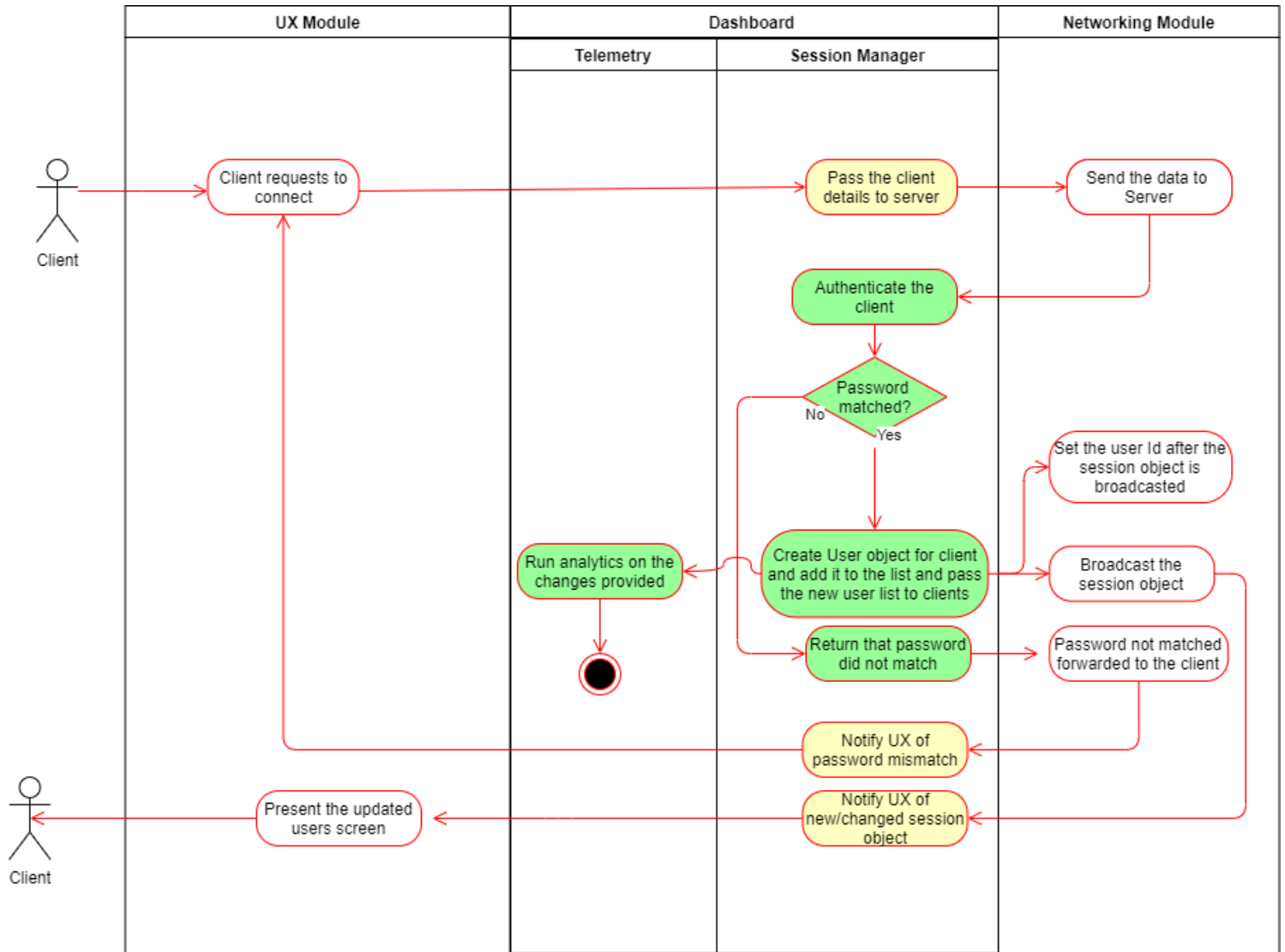


Figure 2: Activity diagram for client join event

### **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.

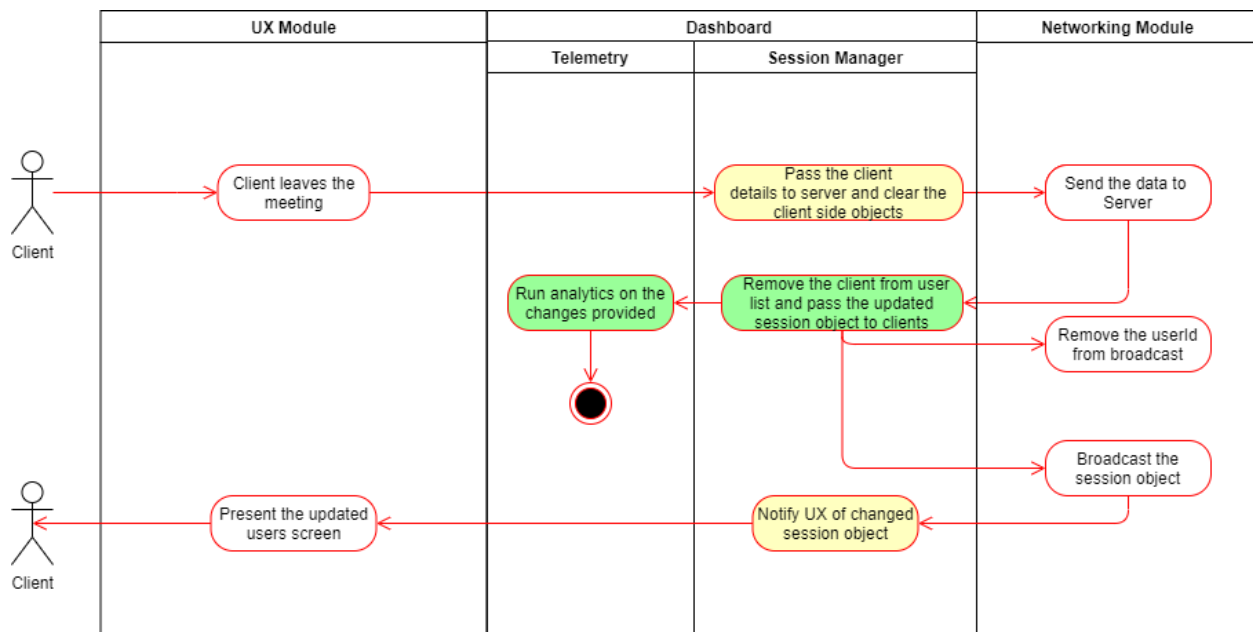


Figure 3: The activity diagram for the event when some client departs

### **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.

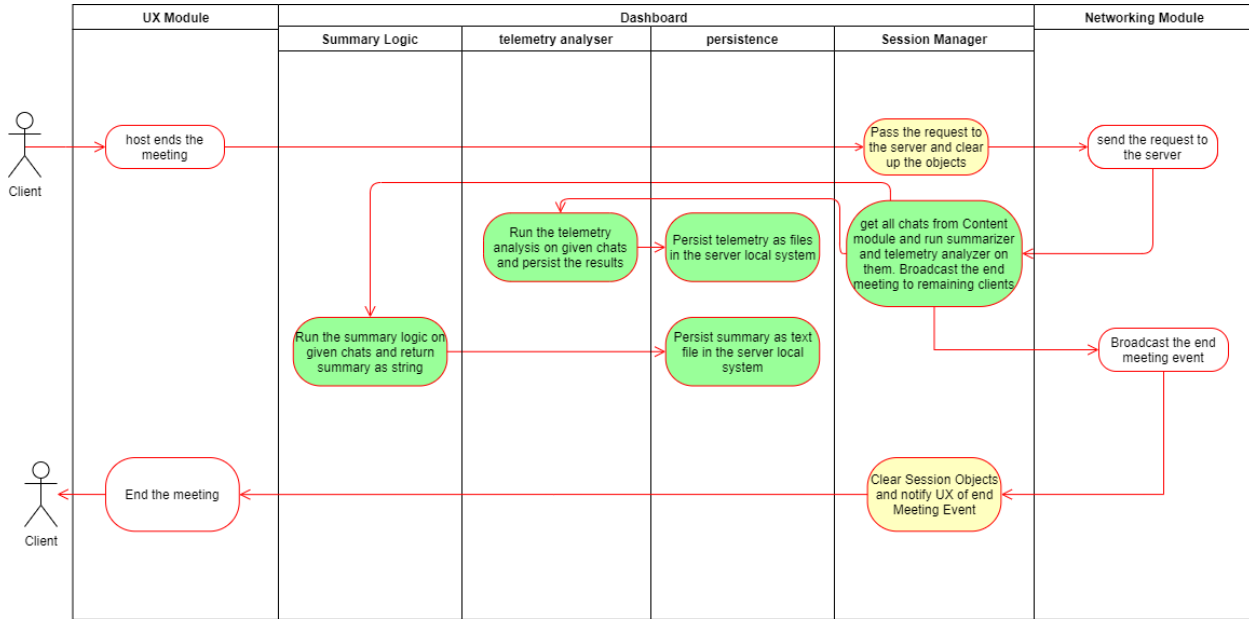


Figure 4: a) Activity Diagram for End Meeting Event.

### **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.

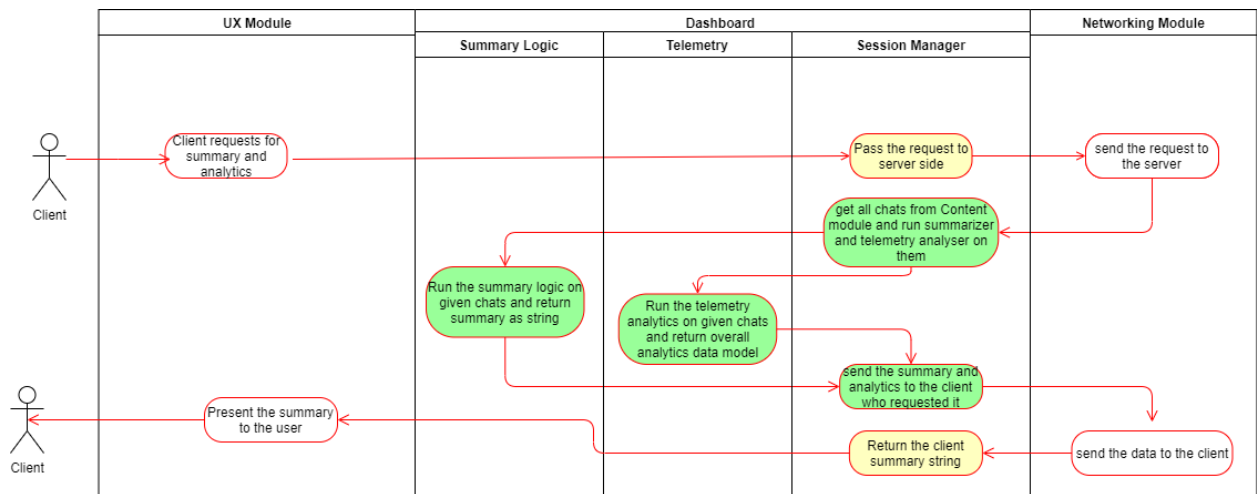


Figure 5: a) Data flow and b) Activity Flow diagrams when some client requests for summary

## 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**

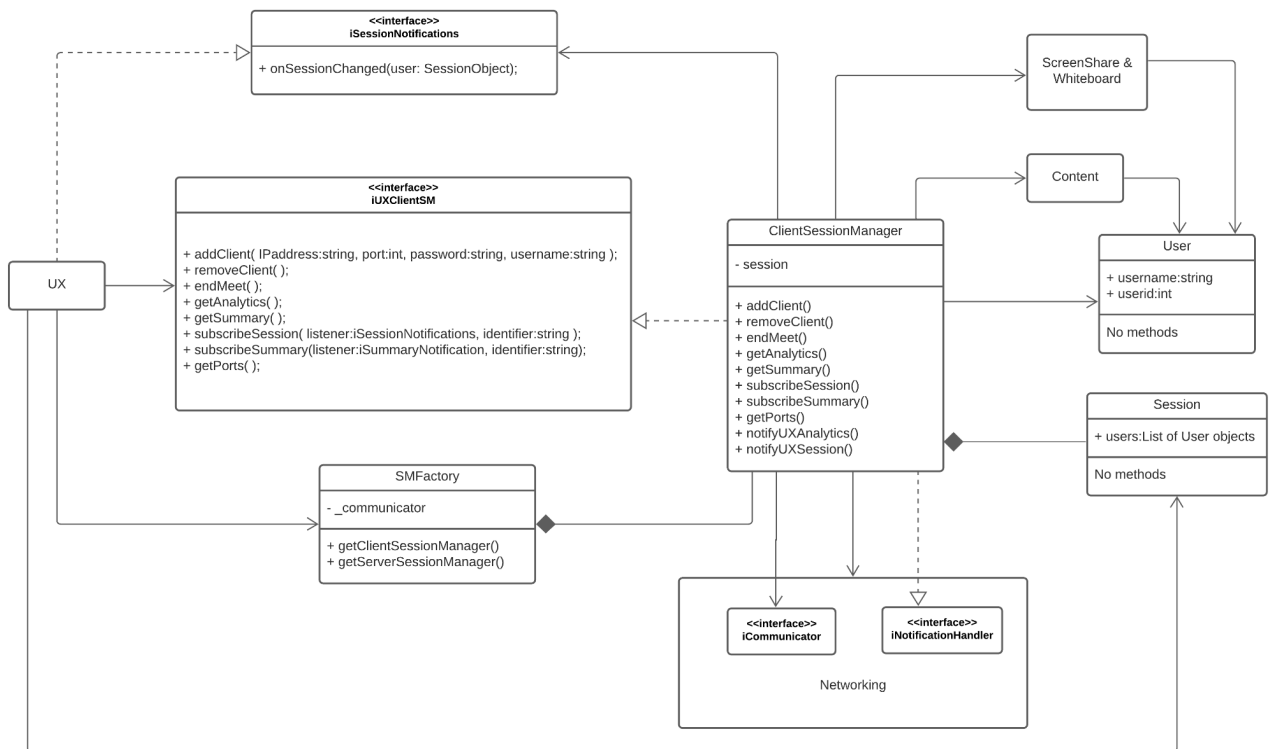
The UX depends on the Session Management module which in turn depends on the Networking Module. The SM (Session Manager) uses the Content module to get the chats and send users to it. Similarly, the Screen-share and Whiteboard module is also used by the SM to send/set users. To achieve this design in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

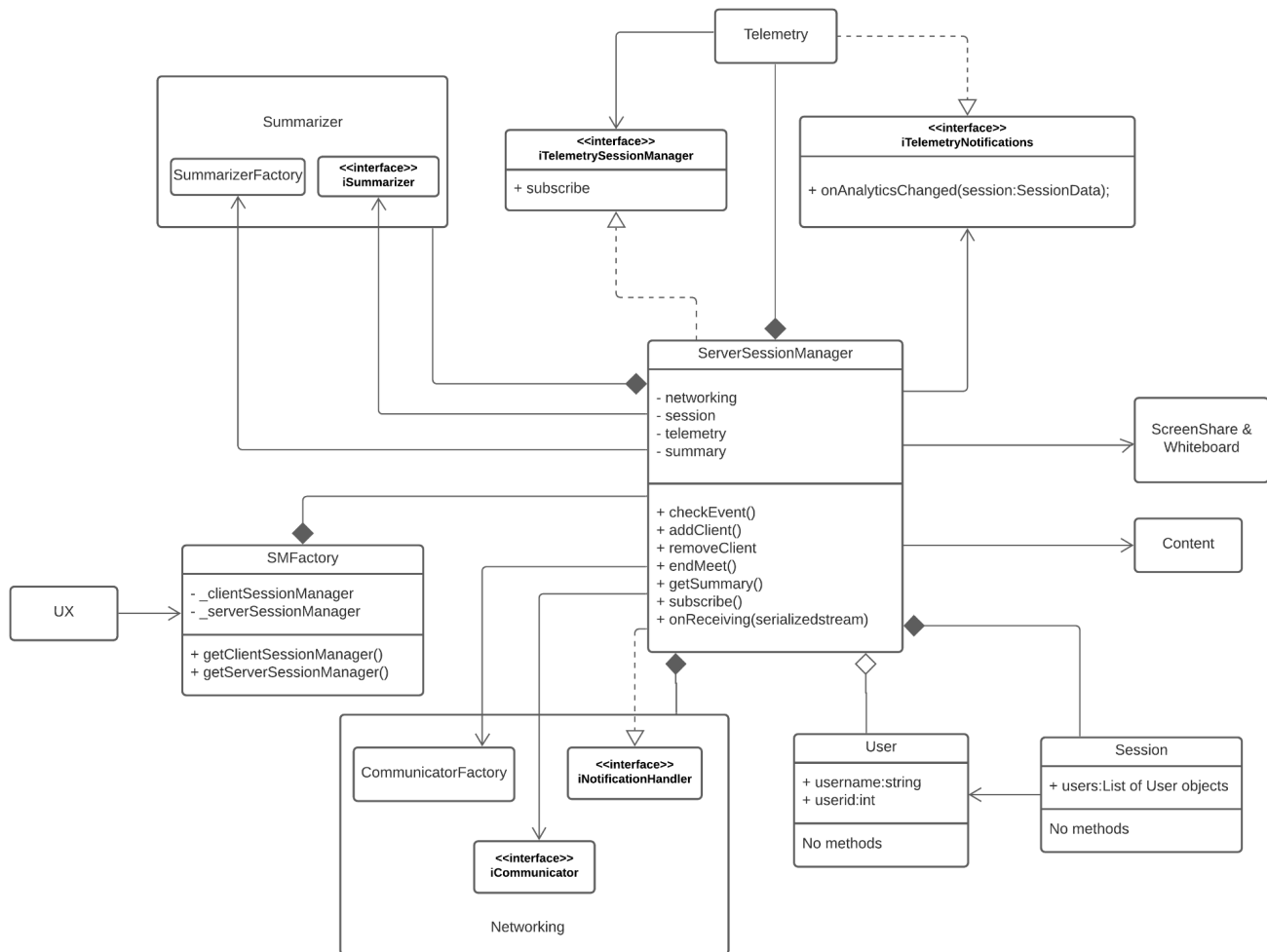
## Class Diagram

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 ipaddress 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

---

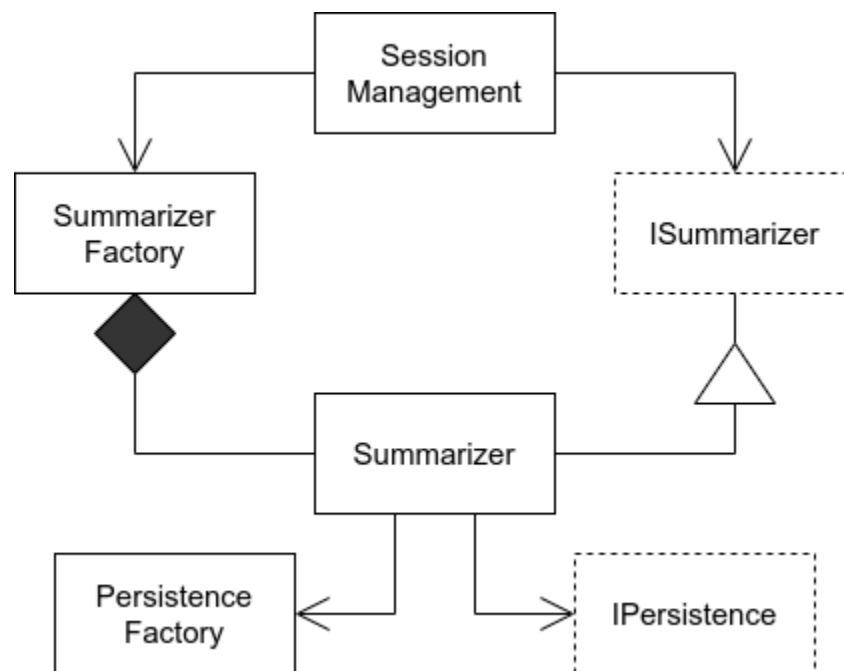
### Overview

In our application, we wish to support summarizing the different properties that would be present such as the chat, images, etc 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);

}
```

## 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.
- The reason for this design choice is that a summarizer if data-driven will learn to perform better and can also summarize the content to provide the relevant information more effectively.
- This summarizer would be created at 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 Machine learning based models under the hood which would be trained on open source chat summary datasets that are available for academic use.
- Machine learning models over the past few years have proven to be extremely helpful in assisting many tasks and automating with human level performance giving almost life-like answers to many query and summarization based tasks with advancements in sequence models like Transformers and attention.
- The generated summary would be used by the session manager to store in the database and retrieved for showing on the dashboard.

## 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 more 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.

## References

Gliwa, B., Mochol, I., Biesek, M., & Wawer, A. (2019). SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization. *Proceedings of the 2nd Workshop on New Frontiers in Summarization*.  
<https://doi.org/10.18653/v1/d19-5409>

## 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:

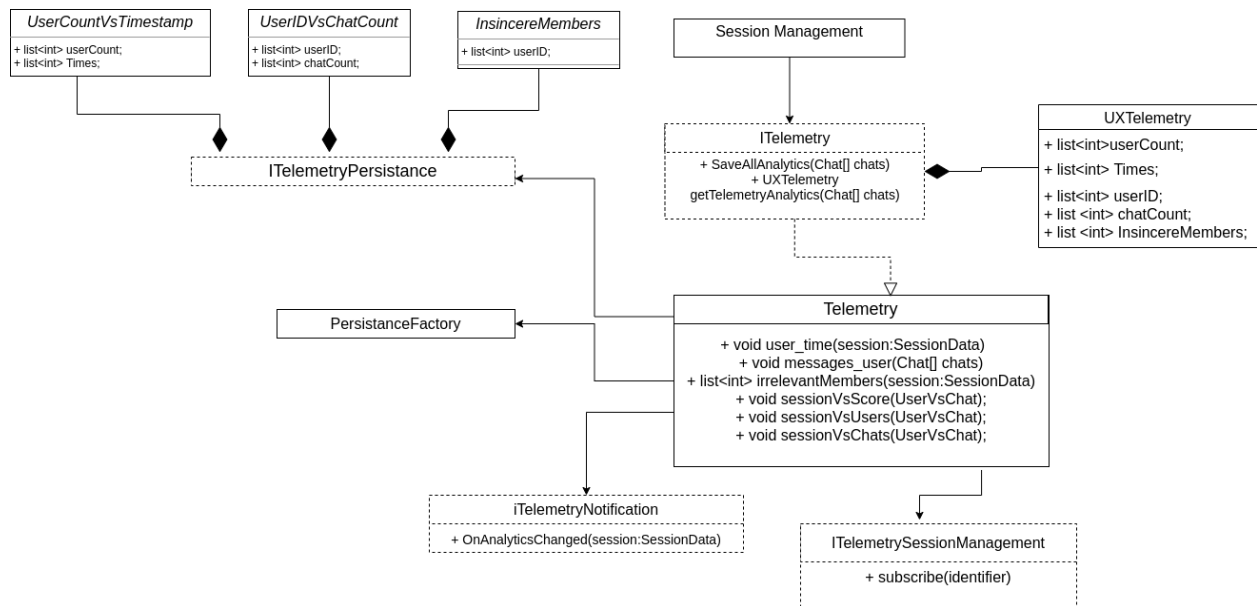
1. Session vs Score: It calculates the score of each session by some logical formula and plots the histogram for it.
2. Session vs users: Plots the histogram for number of users in each session.
3. Session vs chats: Plots the pie chart for number of chats in each session.

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 ITelemetryPersistence, hence, it will be using the ITelemetryPersistence interface. ITelemetryPersistence 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



## **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**

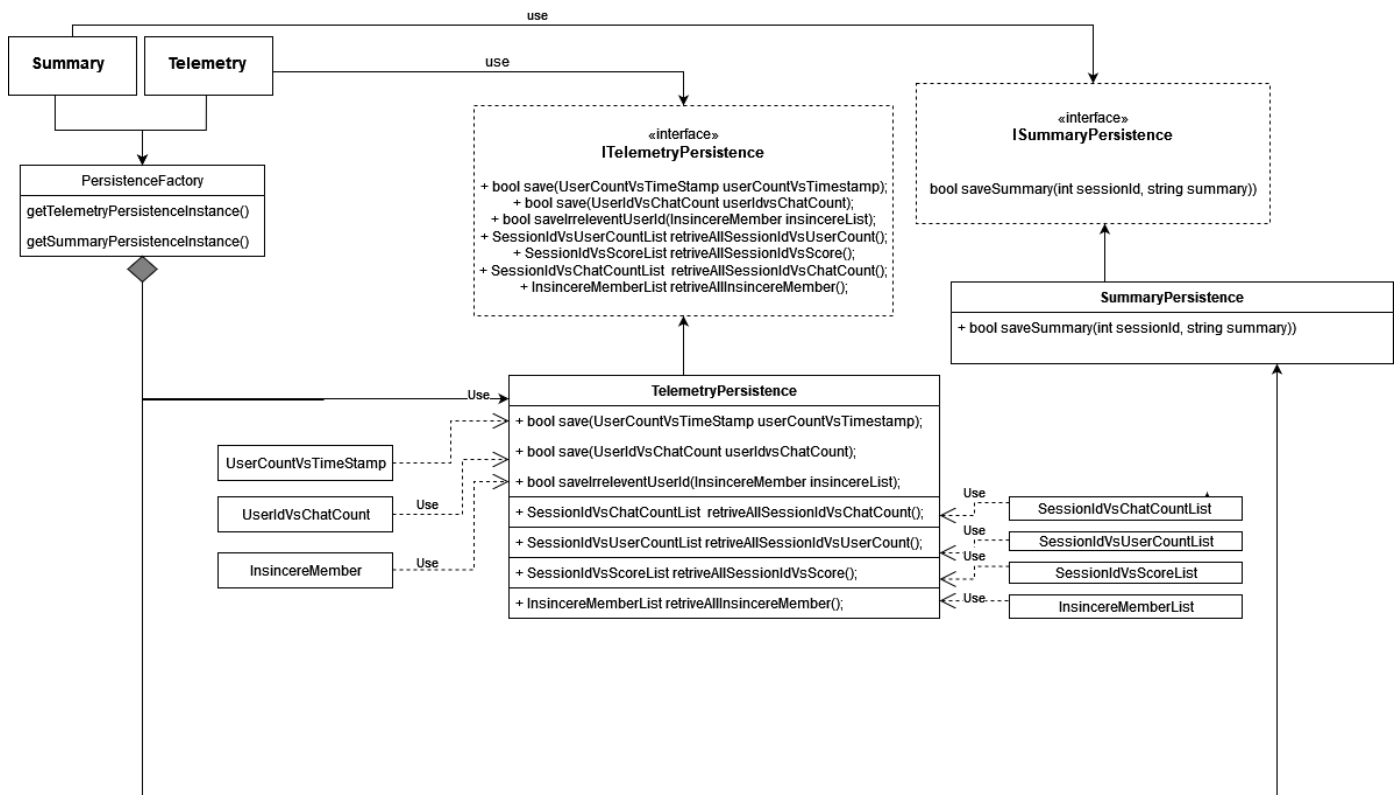
- Stores the Analytics images in the .png format session wise.
- Stores the Analytics data in a meaningful way in .txt files in each session.
- Retrieves the above Server Analytics data and sends it back to the Telemetry module at the end of the meeting.
- Stores the ChatSummary in the .txt format at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



## Interfaces

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

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

//This interface 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 retrieveAllSessionIdVsChatCount();

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

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

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

## 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