CS5617:Software Engineering

Design Specification: Session Management (Dashboard)

Name: Saurabh Kumar, Roll: 111901046
Team Member, Dashboard team

OVERVIEW

- Session management refers explicitly to managing different “sessions,” or periods when the user is logged in and active in the application.
- Users only interact with the UX layer, which is atop the other modules. The Dashboard Module layer, which manages UX's interaction with other modules, is located beneath the UX layer. An essential dashboard submodule is the session manager.
- At the start of the meeting, sessions will be created, and these **entities will be managed during the meeting**.
- Each user participating in the meeting at the time is considered a server session.
- The session manager is responsible to change the mode of the session i.e, **lab mode** and **exam mode**.
- The session manager is also in charge of **requesting statistics** creation and updates from the **telemetry service** and requesting the **meeting summary** from the summarizer.

OBJECTIVES

The session manager's role is as follows:

- The session manager acts as a controller for both the server and the client.
- It ensures that all managers in other modules are initialized with the proper dependencies.
- It serves as an interface for all of the Dashboard module’s submodules.
- It also keeps track of all the session’s users’ information.
- The SM also sends/sets users using the Screen-share and Whiteboard module.

The server and client components of the session manager are separated. The working of the server session manager and client-server manager are as below:

<u>The Server Session Manager:</u>

- The Host chooses the mode: The Server Session Manager provides the option for the host to select the mode of session i.e, Lab Mode or Exam Mode.
- When the host creates the meeting, the Server Session Manager gives the UX the ports and IP address required to join the meeting.
- When a user joins the meeting after the user is successfully authenticated, the server session would get the ID of the user from the UX module. The user will then be added to the active session by the server session manager, who will also notify all clients of the new user through broadcasting. Additionally, it will alert the Telemetry module to provide the statistics in accordance.
- When the user asks for getting the summary, the summarizer module provides the summary to the Server Session Manager. The client-side session manager for that specific user receives the summary when it has been fetched.
- User asks for getting the analytics: The Telemetry module's analytics are retrieved by the Server Session Manager. The analytics are delivered to the client-side session manager of that specific user after being fetched.
- User leaves the meeting: The server Session manager removes the user from that particular session. The telemetry module is notified of this change.
- The last user leaves the meeting: When the Server sessions manager finds that only the last user is present, then it asks the summary and telemetry module to save the summary and analytics. The server UX is then notified of the meet-end event.

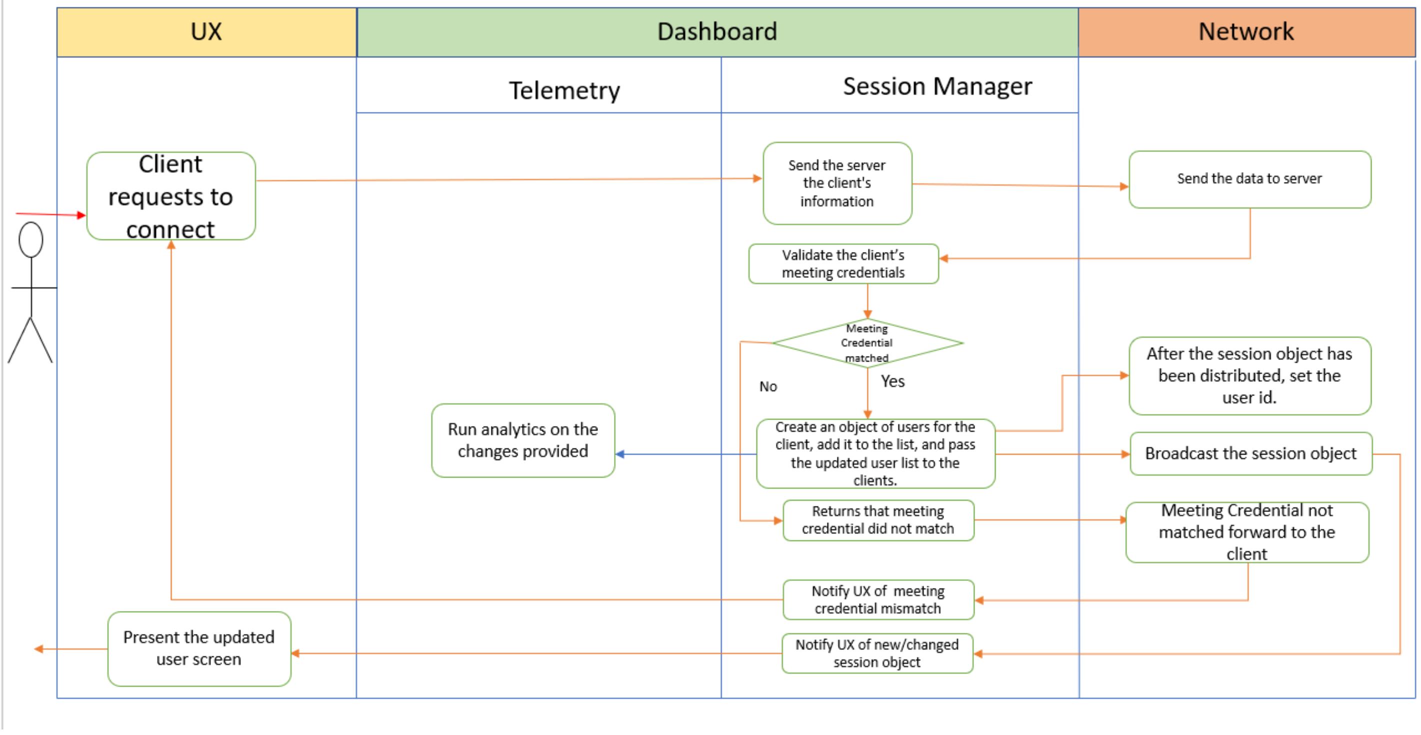
<u>The Client Session Manager:</u>

- User attends the meeting: After confirming the incoming client's credentials, the UX module sends the client's username to the server side. After that, it will update its own session data using the updated session data it received from the server session manager. Then a notification will go to the Client UX module.
- When the user asks for getting the summary, the user's data will be sent to the server side first via the client session manager. The Client UX is then informed about the production of the summary via calling an event after receiving the summary from the server side.
- When a user requests analytics, the Client Session Manager first sends the user's data to the server. The Client UX is then informed of the analytics by triggering an event after receiving the analytics from the server side.

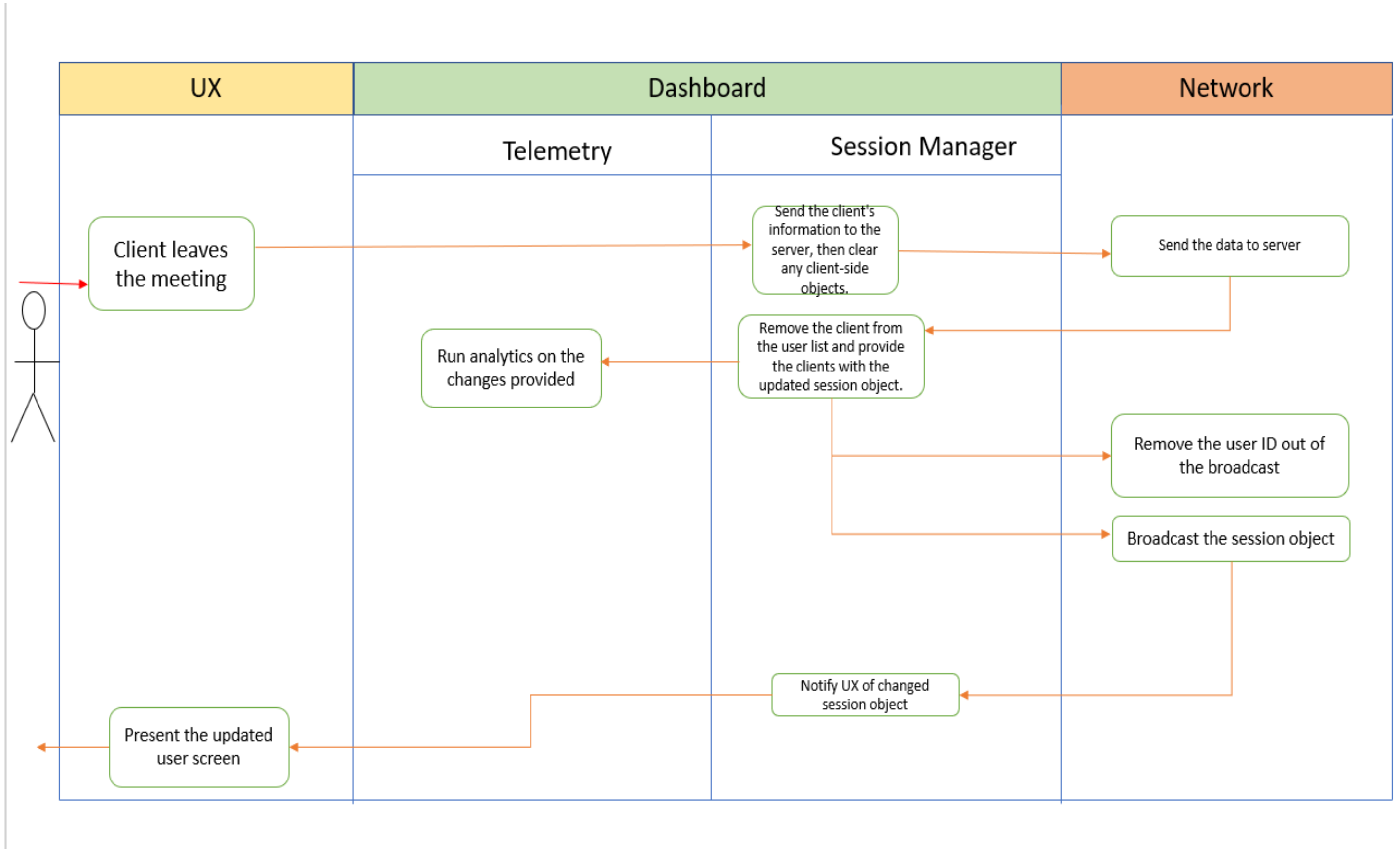
- When the user leaves the meeting, The client's request to terminate the session is communicated to the server session manager by the client session manager. The server session manager permits the client to exit while updating its own session data. The client's user object and session data are set to null after receiving the server's answer.
- When the user leaves the meeting, at last, the Client Session Manager receives the information that the meeting has ended. It then notifies the Client UX about the end of the meeting via invoking an event.

ACTIVITY DIAGRAM:

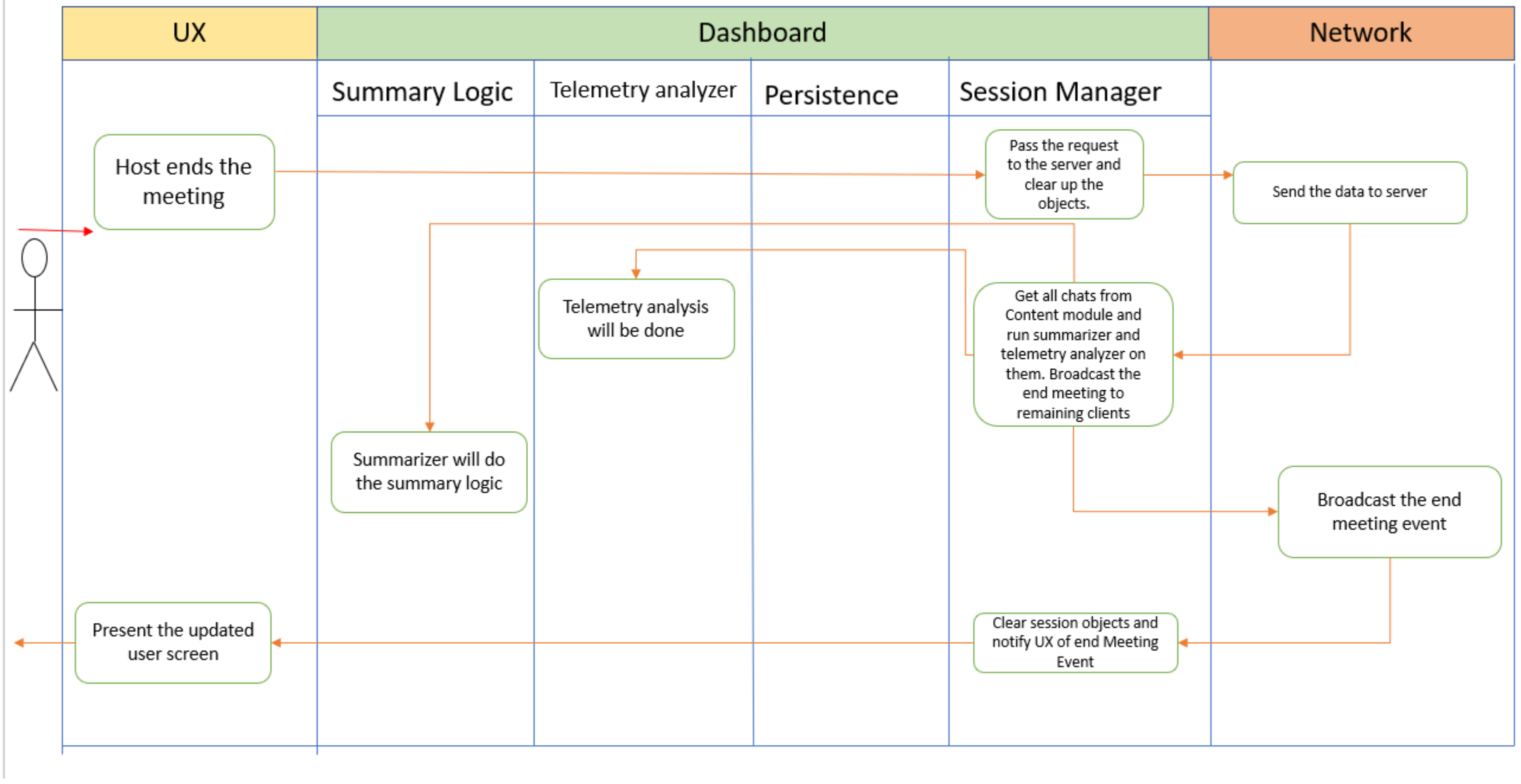
User Joining the meeting: After the user has successfully authenticated from the UX authentication submodule, the UX module will redirect the user to the session manager. The session manager then validates the user whether the user has the correct meeting credentials or not. If the user gets validated, the session manager will create an object of users for the client, then add it to the list and then give the updated user list to the clients. After the session object is broadcasted by the network module, the session manager will notify the UX of a new/changed session object.


UX
Dashboard
Network
Telemetry
Session Manager
Client requests to connect
Send the server the client's information
Send the data to server
Validate the client's meeting credentials
Meeting Credential matched
No
Yes
After the session object has been distributed, set the user id.
Run analytics on the changes provided
Create an object of users for the client, add it to the list, and pass the updated user list to the clients.
Broadcast the session object
Returns that meeting credential did not match
Meeting Credential not matched forward to the client
Notify UX of meeting credential mismatch
Present the updated user screen
Notify UX of new/changed session object

User Leaving the Meeting: When the user leaves the meeting, UX notifies that the Client is leaving the session. The session manager then sends the client's information to the server and then clears any client-side objects. After that, the session manager removes the client from the user list and provides the clients with the updated session object. Then it notifies UX of the changes session object.


UX
Dashboard
Network
Telemetry
Session Manager
Client leaves the meeting
Send the client's information to the server, then clear any client-side objects.
Send the data to server
Remove the client from the user list and provide the clients with the updated session object.
Run analytics on the changes provided
Remove the user ID out of the broadcast
Broadcast the session object
Notify UX of changed session object
Present the updated user screen

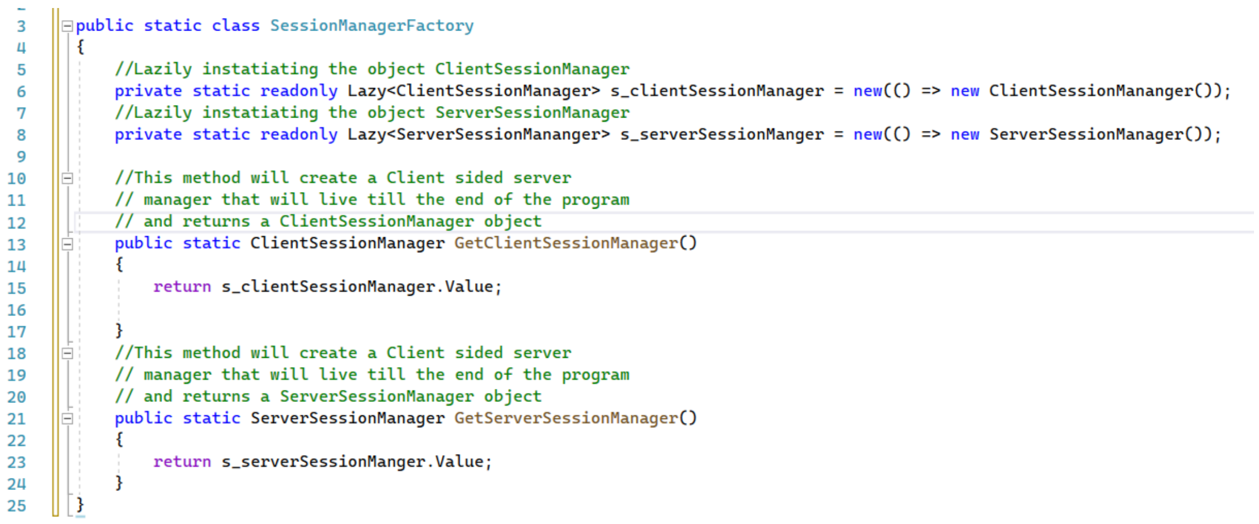
Host ending the meeting: When the host ends the meeting, the session manager should delete the session object on the client side after the meeting. All chats from the Chat module should be collected by the session manager on the server, who should then give them to the telemetry and summary logic submodules for a summary of analyzing the meeting and the information shared therein.


UX
Dashboard
Network
Summary Logic
Telemetry analyzer
Persistence
Session Manager
Host ends the meeting
Pass the request to the server and clear up the objects.
Send the data to server
Telemetry analysis will be done
Get all chats from Content module and run summarizer and telemetry analyzer on them. Broadcast the end meeting to remaining clients
Summarizer will do the summary logic
Broadcast the end meeting event
Present the updated user screen
Clear session objects and notify UX of end Meeting Event

Design Pattern:

- For the session management, we are using the factory design pattern, as, in the Factory pattern, we create objects without exposing the creation logic to the client and refer to newly created objects using a common interface. It allows the sub-classes to choose the type of objects to create. It also promotes the loose-coupling by eliminating the need to bind application-specific classes into the code.
- We are also using the singleton pattern having lazy instantiation, which helps in creating the object only when required. This helps to save memory, as the object is not created at each request. Only a single instance is reused again and again.

Factory for Session Manager



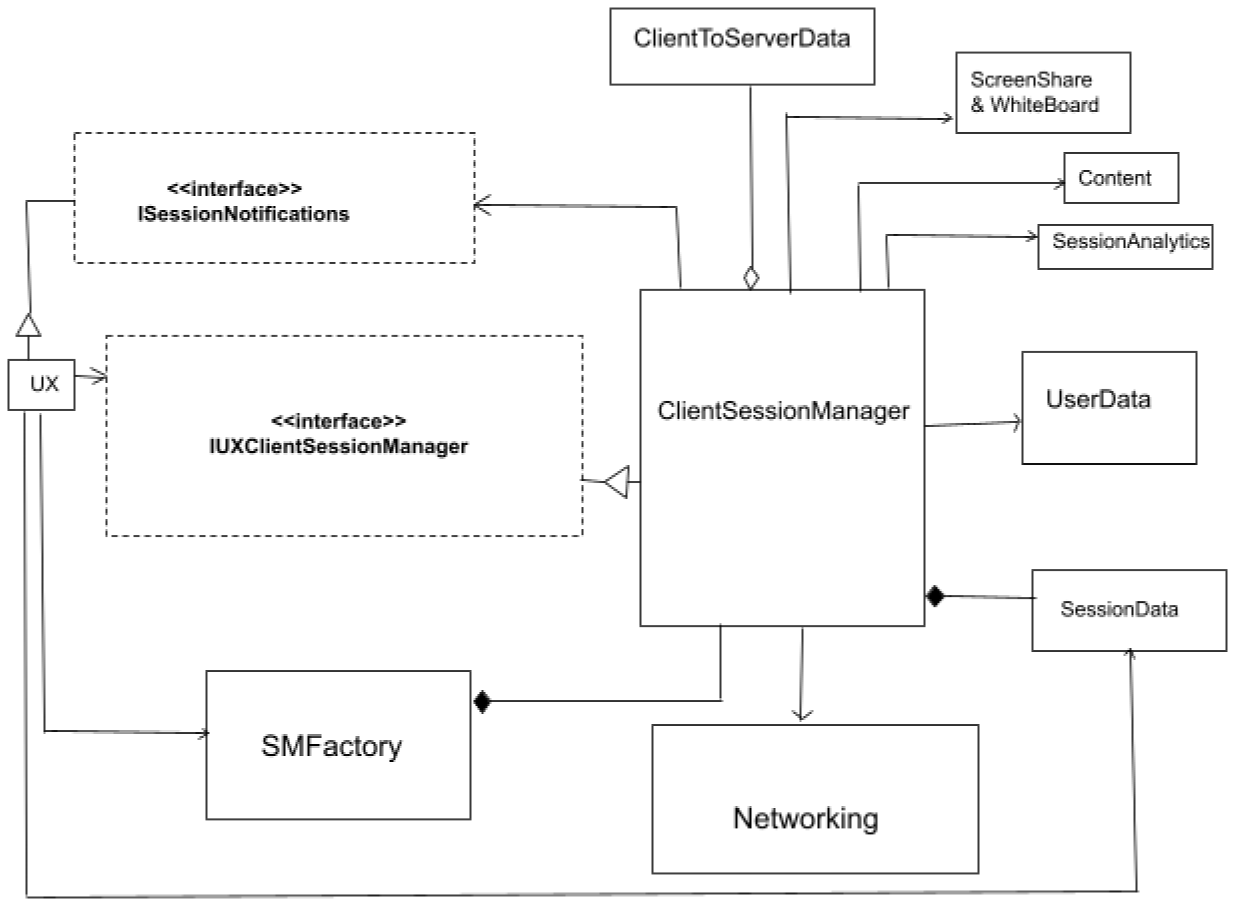
```
public static class SessionManagerFactory
{
    //Lazily instatiating the object ClientSessionManager
    private static readonly Lazy<ClientSessionManager> s_clientSessionManager = new(() => new ClientSessionMananger());
    //Lazily instatiating the object ServerSessionManager
    private static readonly Lazy<ServerSessionMananger> s_serverSessionManger = new(() => new ServerSessionManager());

    //This method will create a Client sided server
    // manager that will live till the end of the program
    // and returns a ClientSessionManager object
    public static ClientSessionManager GetClientSessionManager()
    {
        return s_clientSessionManager.Value;

    }
    //This method will create a Client sided server
    // manager that will live till the end of the program
    // and returns a ServerSessionManager object
    public static ServerSessionManager GetServerSessionManager()
    {
        return s_serverSessionManger.Value;
    }
}
```

In the above, we are also using the singleton pattern having lazy instantiation, which helps in creating the object only when required.

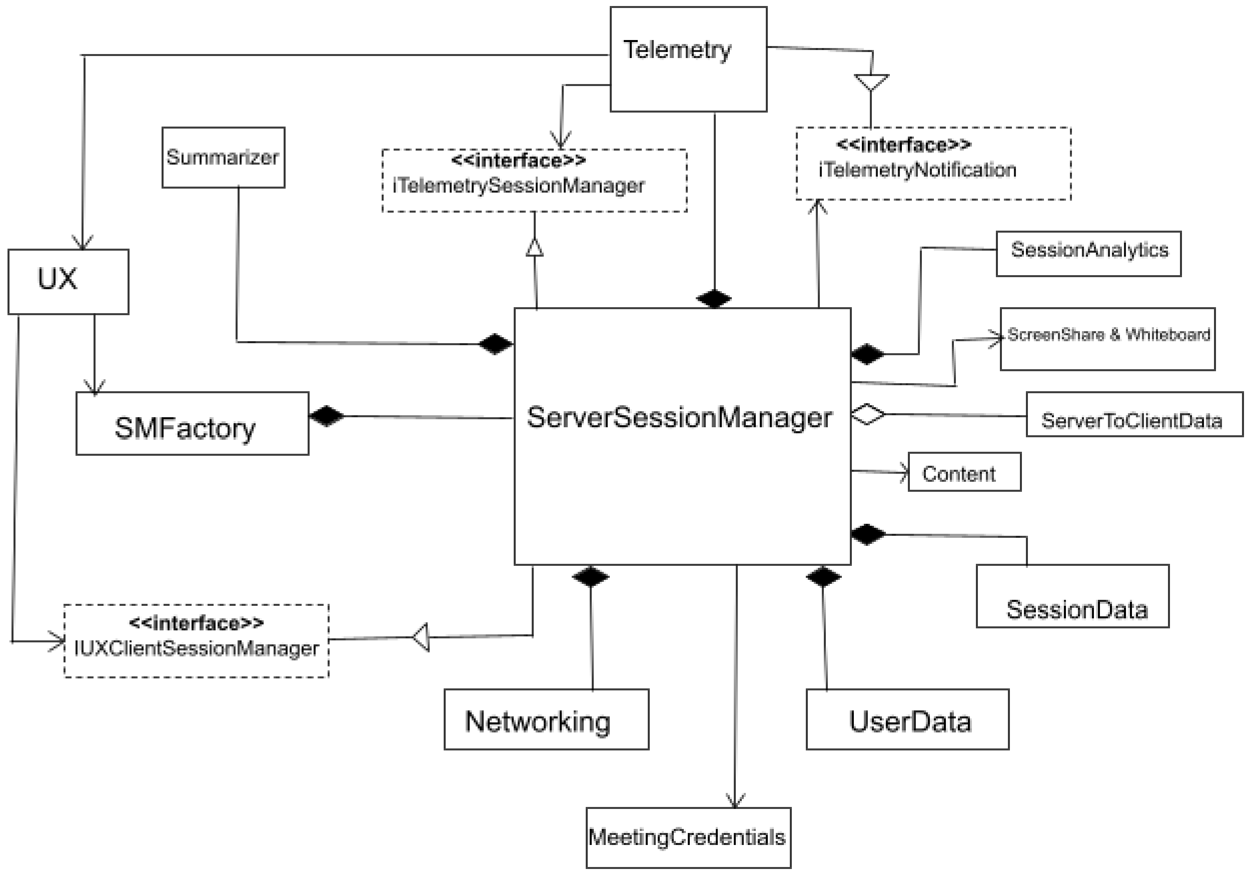
CLASS DIAGRAM:

For simplicity, functions and attributes are not specified in the diagram:

Client Side class diagram of the Session Manager:



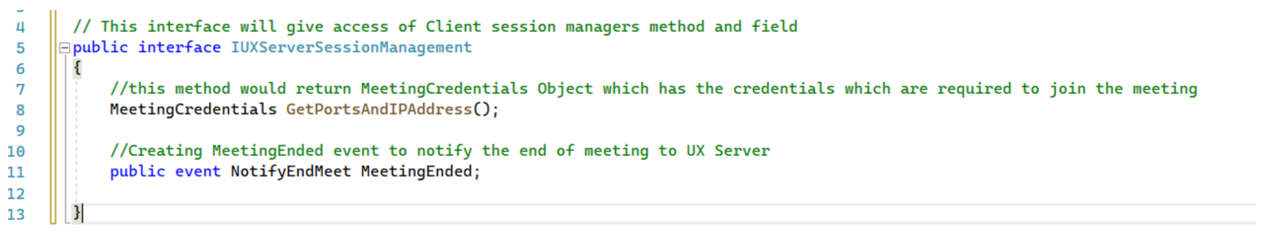
Server Side class diagram of the SessionManger:



INTERFACES:

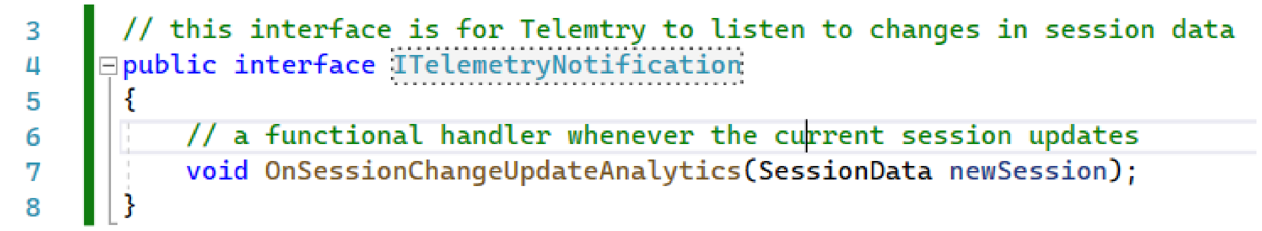
Server Session side Interfaces:

IUXServerSessionManagement



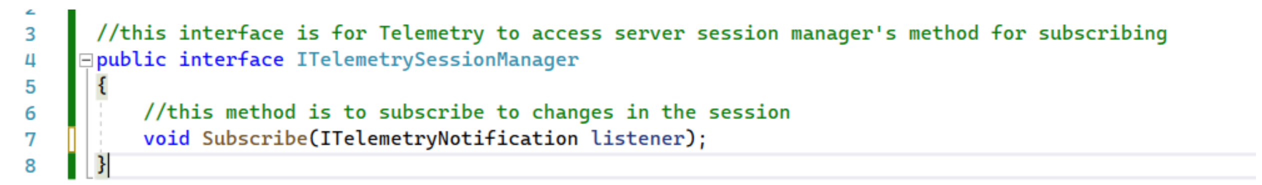
```
// This interface will give access of Client session managers method and field
public interface IUXServerSessionManagement
{
    //this method would return MeetingCredentials Object which has the credentials which are required to join the meeting
    MeetingCredentials GetPortsAndIPAddress();

    //Creating MeetingEnded event to notify the end of meeting to UX Server
    public event NotifyEndMeet MeetingEnded;

}
```

ITelemetryNotification



```
// this interface is for Telemtry to listen to changes in session data
public interface ITelemetryNotification
{
    // a functional handler whenever the current session updates
    void OnSessionChangeUpdateAnalytics(SessionData newSession);
}
```

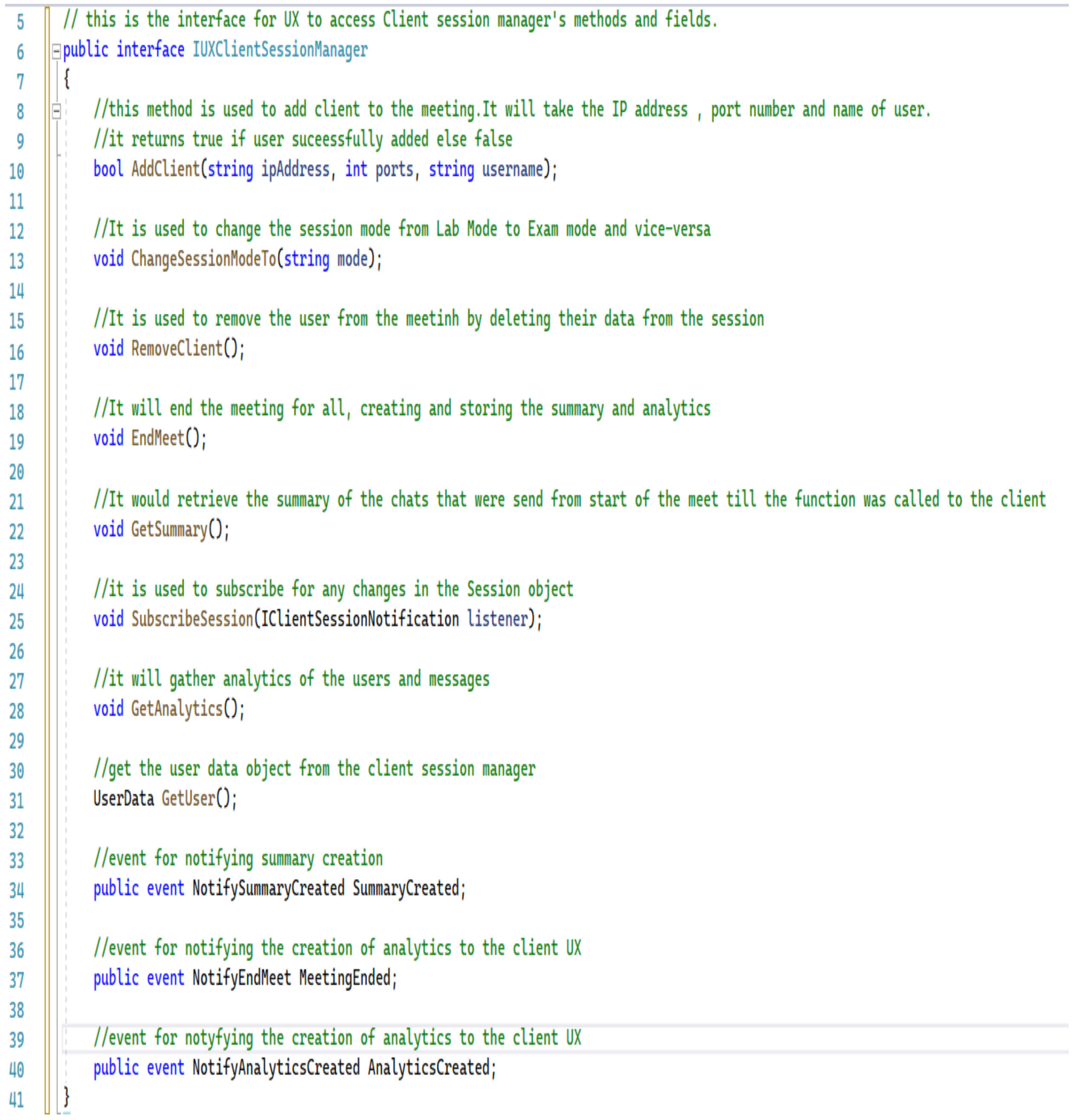
ITelemetrySessionManager



```
//this interface is for Telemetry to access server session manager's method for subscribing
public interface ITelemetrySessionManager
{
    //this method is to subscribe to changes in the session
    void Subscribe(ITelemetryNotification listener);
}
```

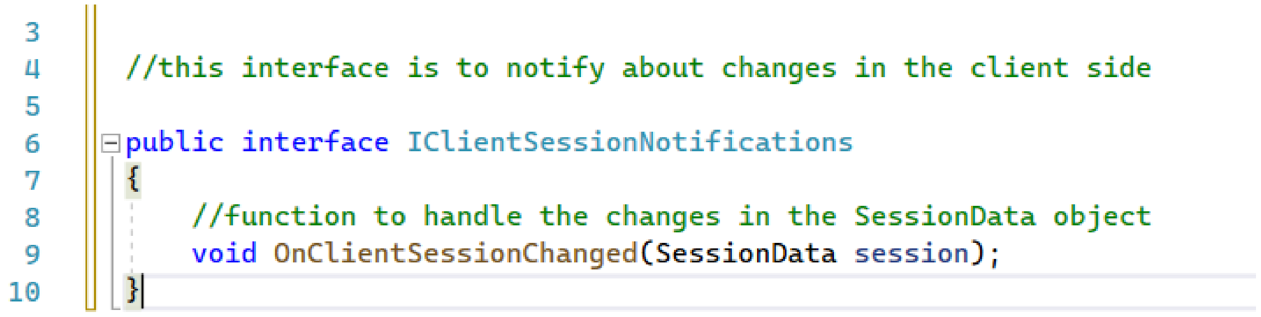
Client Session side Interfaces:

IUXClientSessionManager



```
// this is the interface for UX to access Client session manager's methods and fields.
public interface IUXClientSessionManager
{
    //this method is used to add client to the meeting.It will take the IP address , port number and name of user.
    //it returns true if user suceessfully added else false
    bool AddClient(string ipAddress, int ports, string username);

    //It is used to change the session mode from Lab Mode to Exam mode and vice-versa
    void ChangeSessionModeTo(string mode);

    //It is used to remove the user from the meetinh by deleting their data from the session
    void RemoveClient();

    //It will end the meeting for all, creating and storing the summary and analytics
    void EndMeet();

    //It would retrieve the summary of the chats that were send from start of the meet till the function was called to the client
    void GetSummary();

    //it is used to subscribe for any changes in the Session object
    void SubscribeSession(IClientSessionNotification listener);

    //it will gather analytics of the users and messages
    void GetAnalytics();

    //get the user data object from the client session manager
    UserData GetUser();

    //event for notifying summary creation
    public event NotifySummaryCreated SummaryCreated;

    //event for notifying the creation of analytics to the client UX
    public event NotifyEndMeet MeetingEnded;

    //event for notyfying the creation of analytics to the client UX
    public event NotifyAnalyticsCreated AnalyticsCreated;
}
```

IClientSessionNotifications



```

//this interface is to notify about changes in the client side

public interface IClientSessionNotifications
{
    //function to handle the changes in the SessionData object
    void OnClientSessionChanged(SessionData session);
}
```

DESIGN ANALYSIS

The Session Management module depends on the Networking Module for the UX. The Content module is used by the Session Manager (SM) to retrieve the chats and direct users there. Similar to this, the SM also sends/sets users using the Screen-share and Whiteboard module.

The following actions can be taken to execute this design:

- On the server side, the Session Manager(SM) will keep track of a session object that contains a list of every user present at the meeting.
- The SM will give the UX a factory via which the UX can build client-side and server-side Session manager objects, that will last until the end of the meeting.
- The session management module will include an interface that the UX module will use with the help of defined functionalities.
- The session management module will also include listeners for various subscription types to inform the UX of any changes.
- The SM will receive a factory from the networking module so it can create a networking object. The Session Management Module will use an interface that the networking module will provide.
- The Networking module will provide listener(s), to which the Session management module will subscribe.
- The Content module will also create a user interface that the SM may use to configure people or access conversations on chat. The same goes for Screen-share and Whiteboard modules. The screen-share module will send the info about which users are sharing the screen.
- The SM will also include interfaces from the Telemetry and Summarizer submodules to fetch or save the summary and analytics, respectively. The

respective factories of these submodules will be used to construct these objects. Additionally, the SM will supply the Telemetry with a listener so that it can be informed whenever the analytics change.

- To have the communication between the server and client session manager they will use the ClientToServerData object to send information about the type of event and the client who requested it. Similar to this, the ServerToClientData object is used to send data like summary data, session data, etc.
- Here we are using Factory pattern, in which we create objects without exposing the creation logic to the client and refer to newly created objects using a common interface. It allows the sub-classes to choose the type of objects to create. It also promotes the loose-coupling by eliminating the need to bind application-specific classes into the code.
- In order to save memory, we are using the singleton lazy instantiation, so that objects are created only when needed.

SUMMARY & CONCLUSION:

The session manager acts as a controller on the server as well as on the client side. It maintains the details of all the users in the session. It will create sessions at the beginning of the meeting and manage these entities during an ongoing meeting. It acts as an interface between all the submodules in the Dashboard module. Whenever a user is sharing the screen, using the whiteboard, or messaging in the chat the session manager will get the info from the corresponding module and update the user’s data accordingly. The Session Manager also handles the dynamic change in the session mode i.e, from Lab Mode to Exam Mode and vice-versa. The session manager is responsible for asking the telemetry service to create/update statistics and the summariser to fetch the summary of the meeting.