CS5617 Software Engineering

DESIGN SPECIFICATION

Prepared by **Rupesh Kumar , 111901045, Dashboard Team.**

DASHBOARD UX AND TELEMETRY SUBMODULE

DASHBOARD UX

OVERVIEW

Dashboard is the most important part of the UX of the whole application. Here the dashboard will show the details about the current meeting like the participants list in the meeting and also it will show the telemetric analysis. The telemetric analysis will mainly show the details about the ongoing meeting , load on the server and network. The dashboard will show the total number of participants, multiple graphs about the number of users, number of chats and so on.

OBJECTIVE

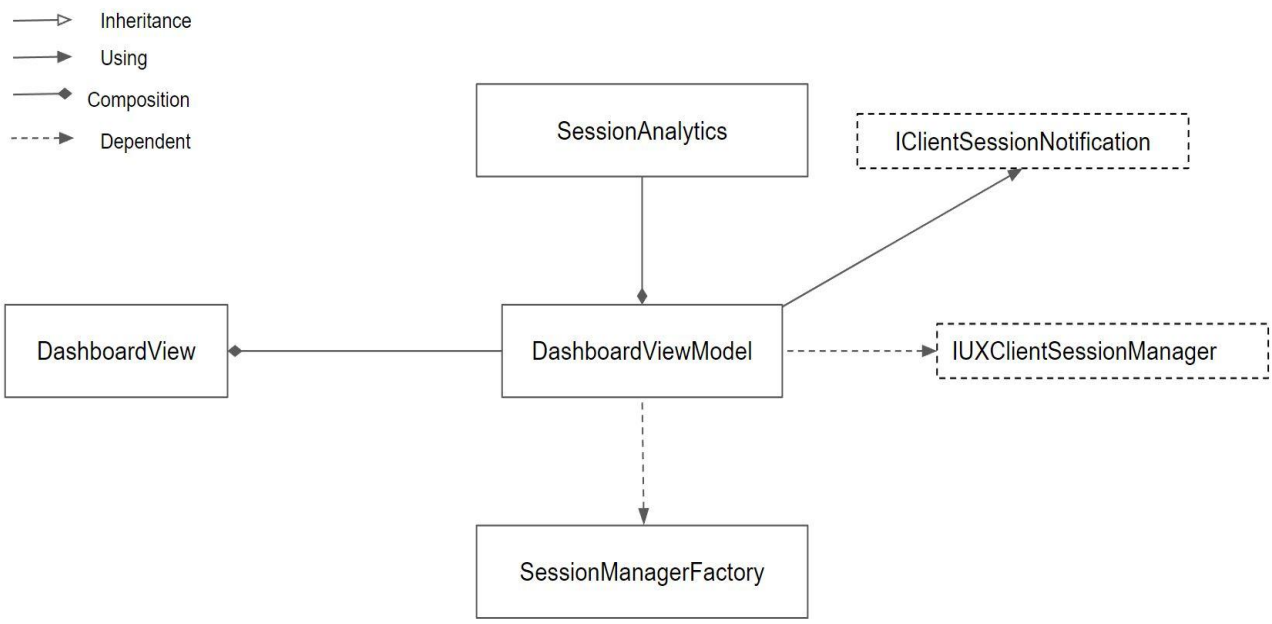
The main objectives are as follows:

- To design the UX for the dashboard module in such a way that it properly shows the graphs, histograms and pie charts about the telemetric data.
- To be able to show the detailed analysis of the telemetric data by showing the usercountVsTimeStamp, chatcountVsUser, totalPacketsVsModule, AverageLatencyVsModule, total number of active users, total number attentive users currently and total number of inattentive users.
- To be able to show the participants list along with information about who all are sharing the screen in the meet.

- To provide an option to the user to refresh the analytics or show the analytics at any time.

Class/UML Diagram

Lets see the class diagram and look at the dependency of the dashboard UX.



Based on the above diagram lets see the design and approach.

DESIGN & APPROACH

- **DashboardView** acts as the code-behind for the XAML file. The bindings like **onRefreshButton** will be defined in this class. This basically handles the structure, layout and orientation of the dashboard screen. The **DashboardView** is instantiating the **DashboardViewModel.**

- **DashboardViewModel** contains the logic to fetch the data from the telemetry , summary module and store the data in proper format to be able to render it on

the view. It is dependent on the interface **IUXClientSessionManager** to get the telemetry data, summary data from the corresponding submodules.

- **IClientSessionNotification** this interface will be inherited by the **DashboardViewModel** to be able to notify the dashboard UX module about the change in the client side session data change.

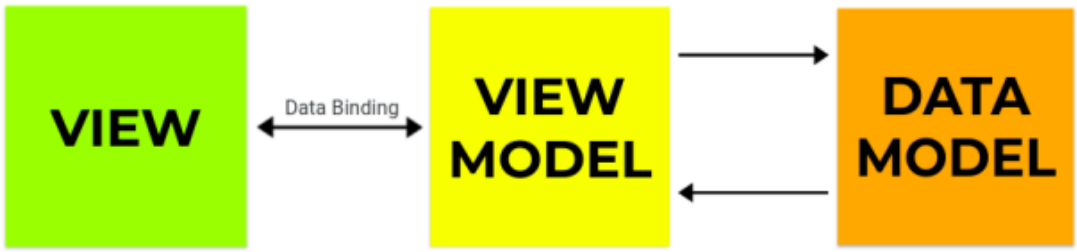
- Whenever the dashboard tab is opened then it will automatically calculate the telemetry analysis data and show the corresponding information and graphs to the UI.
- There will also be a **refresh button** in the dashboard page. On clicking the refresh button option the event listener **OnRefreshButton()** will be activated in the **DashboardView** and this will call the **DashboardViewModel** to get the updated telemetry data by calling **GetAnalytics().** This will update the dashboard telemetric analysis. Also to get the summary at the end **GetSummary()** will be called and shown to the user.
- We will show the **UsersCountVsTimeStamp, ChatCountVsUser, TotalPacketsVsModule, AverageLatencyVsModule, TotalNumberofUsers, TotalChatMessages, AttentiveStudents, InAttentiveStudents.**

DESIGN PATTERNS USED

For the Dashboard UX we are using the MVVM design pattern. MVVM helps in keeping the logic, view separate. This also helps in the unit testing of the applications, helps in having the logical data binding and better reusability of the codes.
In this there are three parts :
View ⇒ This mainly is the interface or UI of the dashboard. This is mainly rendering the XAML file.

ViewModel ⇒ This is basically the place where the binding occurs. The request or the events from the View will be binded here and the also it is connected to the data model where the actual logic or computation will be done.

Data model ⇒ This is the place where actual business logic or the algorithms are run.



- We have to use the **publisher-subscriber** in the **IClientSessionNotification** in which whenever the session changes then the UX module will be notified about the change in session data.
- We are using a singleton **design pattern** for **SessionAnalytics** as we can use only the single instance and do the required work.
- **Factory design pattern** is used in order to get the abstraction from instantiating the instance of **SessionFactory** client side or server side.

DESIGN DECISION MADE (Based on analysis of performance, efficiency,overload)

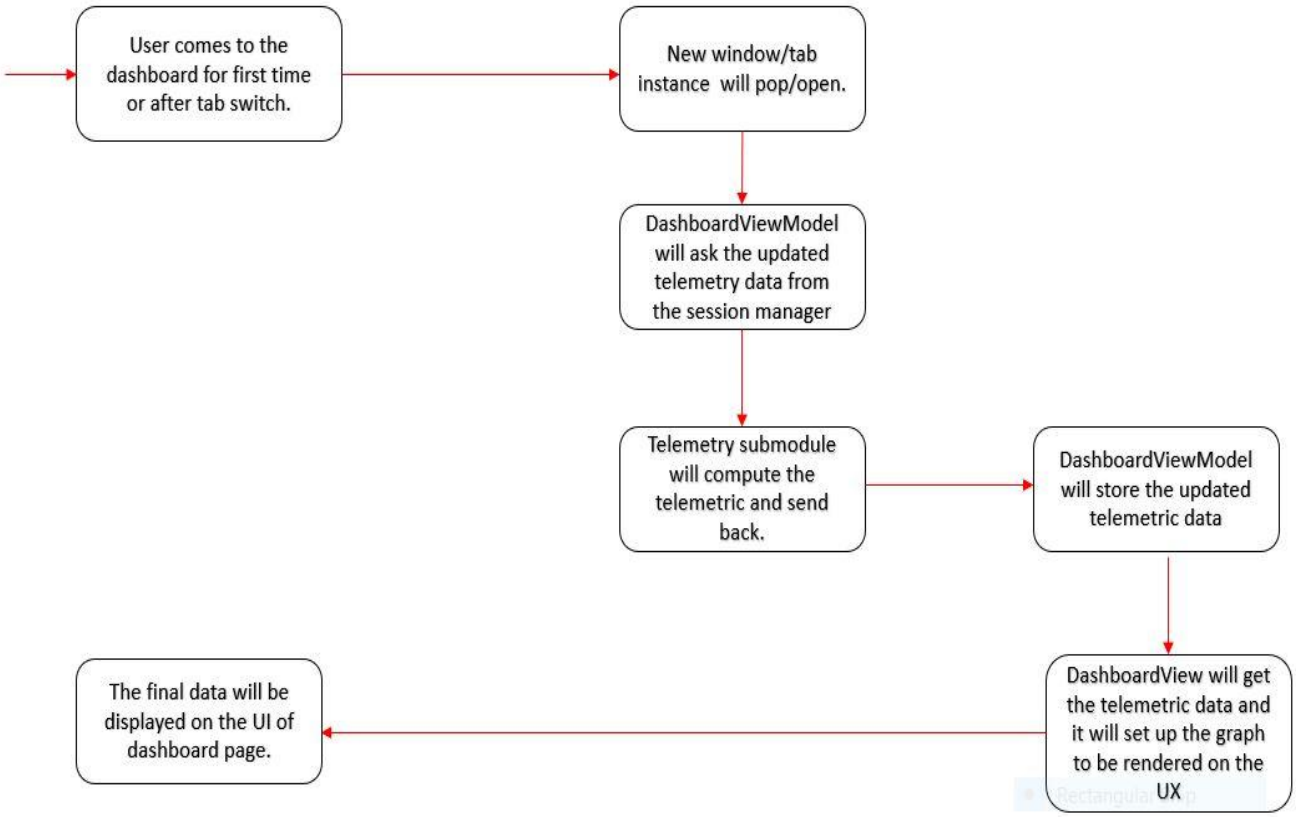
- We are not updating the telemetry and summary module every time or every interval as this will be a big overload for the application and also to the network module. We are updating the statistics whenever the user comes to the dashboard after a tab switch.
- Also If the user remains on the dashboard tab then also there will not be automatic updation of the data. As this will be a bottleneck for the network and performance of the application. So we have a refresh button. The telemetry data will be updated whenever the user clicks the refresh button.

ACTIVITY DIAGRAM

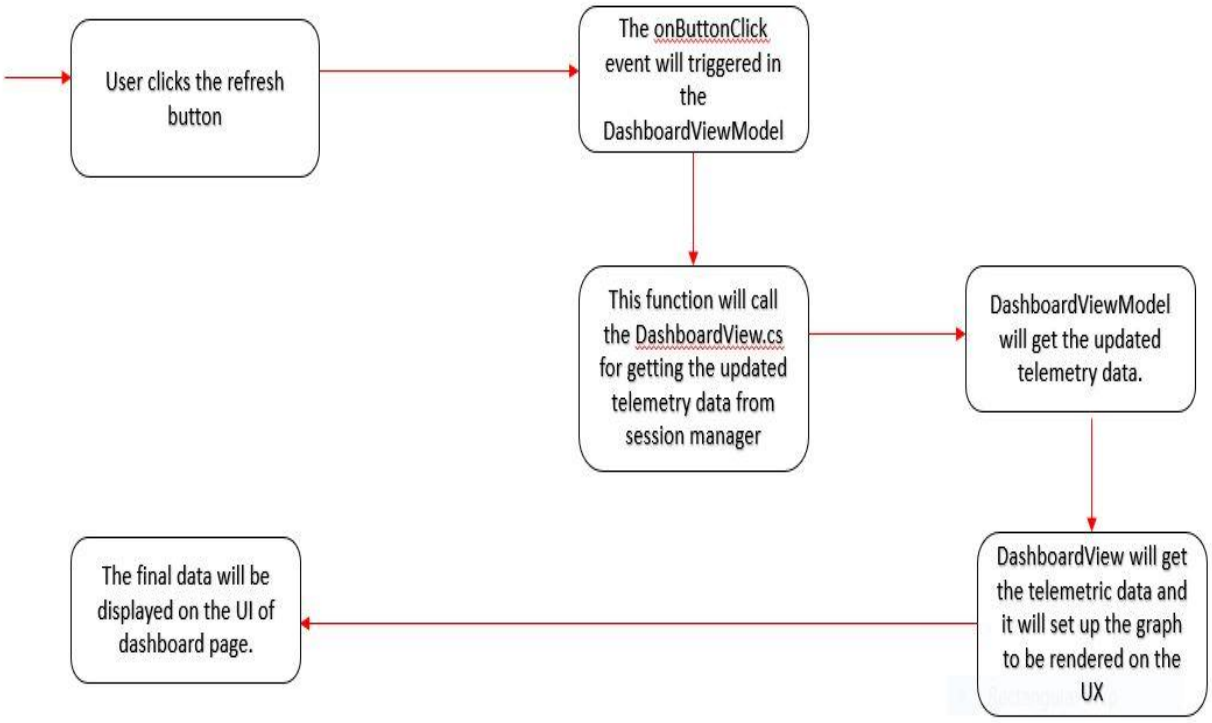
The activity diagram is as follows.

- Whenever the user comes on the dashboard for the first time or after a tab switch then **DashboardViewModel** will ask for the updated telemetric data and

this data will be transferred to the **DashboardView** and this will give the rendering logic for the UI.



- When the user clicks on the refresh button then the on button click event will be triggered and then it will get the updated telemetry data from the session manager. Then it stores in proper format and shows to the UI of the dashboard.



INTERFACES & CLASSES

In this the list of classes are interfaces being used are as follows

1. **IClientSessionManager** ⇒ this is the interface which this has to subscribe in order to get the notification whenever the session data at the client side changes.

2. **DashboardViewModel** ⇒ this is a class which contains the logic to fetch the updated telemetry data from the session manager.

3. **DashboardView** ⇒ this is a class .xaml.cs which contains the event handlers like **OnRefreshButton().** This handles the event and updates the UI.

4. **IUXClientSessionManager** ⇒ this is an interface for the UX of the dashboard to be able to fetch the telemetric data whenever it is needed.
5. **SessionAnalytics** ⇒ this class is used to store the telemetry data and make an object out of it.
6. **SessionManagerFactory** ⇒ this is the factory to get the client side session manager.

TELEMETRY SUBMODULE

OVERVIEW

Telemetry is an important part of the dashboard module. Telemetry means to provide deep analytics and insights about the software application. This module gives analysis about the data and hence provides insights to the user.
In this case the telemetry will provide the deep analysis and insights of the session data of both the lab mode or exam mode. The main function of the telemetry submodule is to give analytics, statistics and insights of each session. Whenever the session is updated either due to entry of a new user or due to exit of a user the telemetry will update itself and provide the updated insights.
It also provides the session analytics and small summary about the session after the session ends.
All the details about the telemetry analysis will be shown to the end user on the dashboard using histograms, pie charts and graphs.

OBJECTIVE

- To give an interface to the server side session manager to be able to fetch/get and save the telemetry data and insights of a particular session.
- To calculate the new telemetry analytics whenever the new user enters into the session or current user leaves the session.
- To provide the telemetry data about:
    - **a.** the time for which each user remains active in the session. This will distinguish the active users vs idle users.
    - **b.** Total number of chats per user in the session.
    - **c.** Total number of users in the session at a given time.
    - **d.** Total number of packets being transferred via networking of each module.
    - **e.** Average latency of each module.
- To be able to show the telemetry analytics on the UX with graphs and histograms using the persistence module.
- To persist or save the telemetry data using the persistence submodule on the server or host computer in a file.
- To provide a small summary at the end of the session.
- To be able to run in the background without blocking the other processes.

DESIGN AND APPROACH

Basically the telemetry module will be dependent on the server side session manager, networking and the persistent module to be able to function properly.

- The telemetry module will inherit the interface named **ITelemetry.** Since the server side session manager needs the option to get and save the telemetry analytics we will need two functions **SaveTelemetryAnalytics** and **GetTelemetryAnalytics** inside this interface which will be implemented by the telemetry module.
- There is a **TelemetryFactory** which helps in instantiating the telemetry module. This uses the factory design pattern.
- The telemetry module will have to subscribe to the server side session manager so that the telemetry data gets updated every time the session data on the server side changes either due to removal of client or arrival of new client.
- We will have **SessionAnalytics** which stores the information for telemetry analytics. The session analytics will be stored after the meeting ends. In this class we have to store the
    - **ChatCountPerUser** ⇒ how many chat messages has been sent by a user
    - **AttentiveMembers** ⇒ Number of students who were present above a threshold time
    - **NonAttentiveMembers** ⇒ Number students who left early
    - **UserCountVsTime** ⇒ Total number of users attending the session at a given time
    - **TotalPacketsVsModule** ⇒ Total number of packets being transferred by each module in a session.
    - **AverageLatencyVsModule** ⇒ Stores the average latency corresponding to each module in a session.
- We also have to store the session summary. For this we have a class **SessionSummary** which will store the
    - **chatCount** ⇒ total number of chats that has been done in the session
    - **userCount** ⇒ total number of users in the session.
    - **sessionScore** ⇒ total score of the session. This can be calculated using chatCount and userCount in the session.
- The telemetry module needs another interface **ITelemetryNotification** to subscribe to the **OnSessionChangeUpdateAnalytics().** This function will be triggered to those who have subscribed to this whenever the session changes on

the server side. So this basically works using the publisher-subscriber design pattern.

- The telemetry module has the interface to subscribe to the on session change event.
- For getting the statistics from the networking module we have an interface named **ITelemetryNetworking** interface that will be implemented by the networking team in which we can get the information about the **TotalPacketsVsModule,** and **AverageLatencyVsModule.**

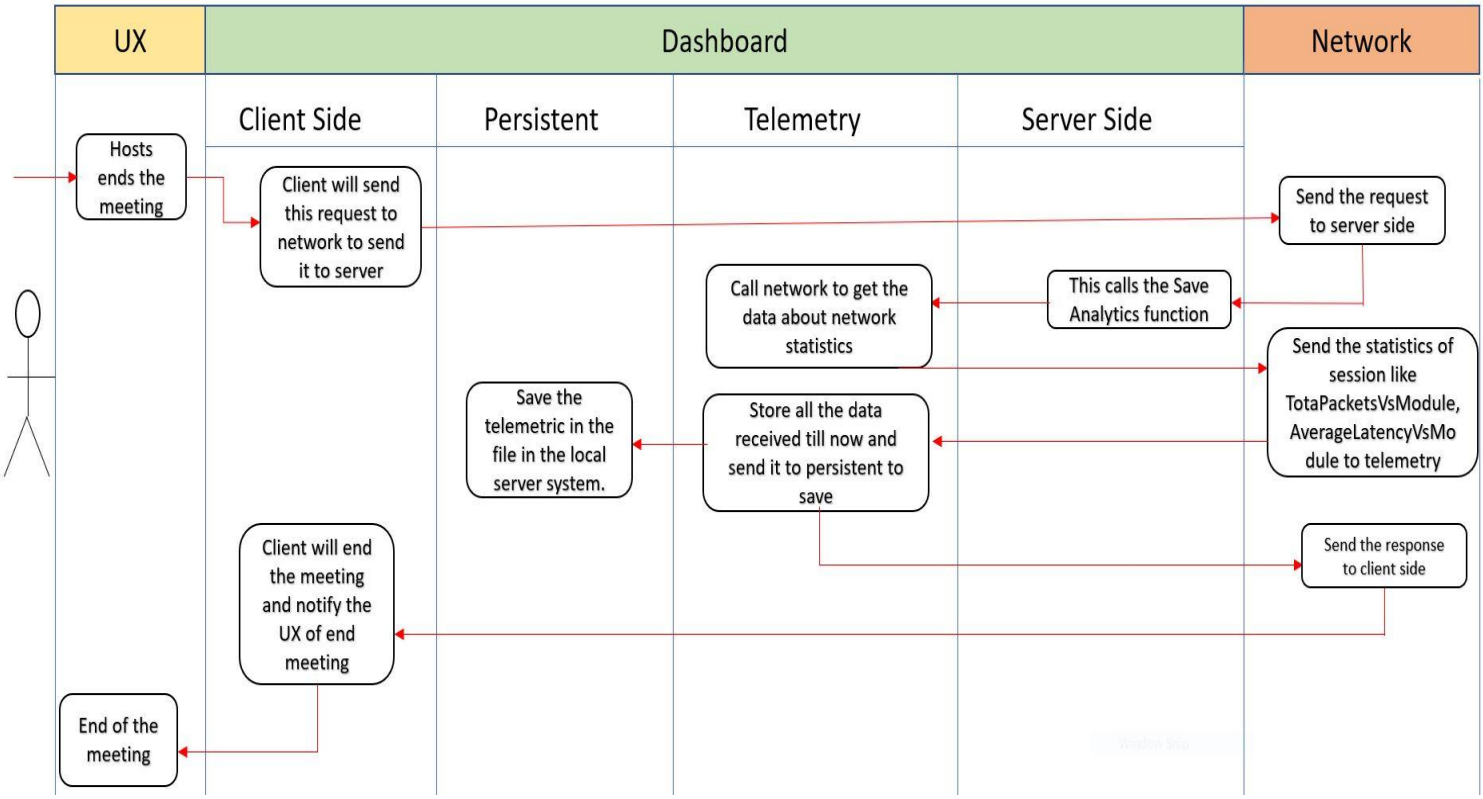
ACTIVITY DIAGRAM

Let's look at the activity diagram for some special events for the telemetry module.

TELEMETRY AT THE END OF SESSION

When the host clicks on the end meet button then this request will go to the client side to end the meeting. The client side then sends the request to the server side via a network module with an event name as **EndMeet** which will start the end meet procedure in which it will call the **SaveAnalytics** function to get the analytics and save it. Then the persistence module will save the telemetry analytics into a file.

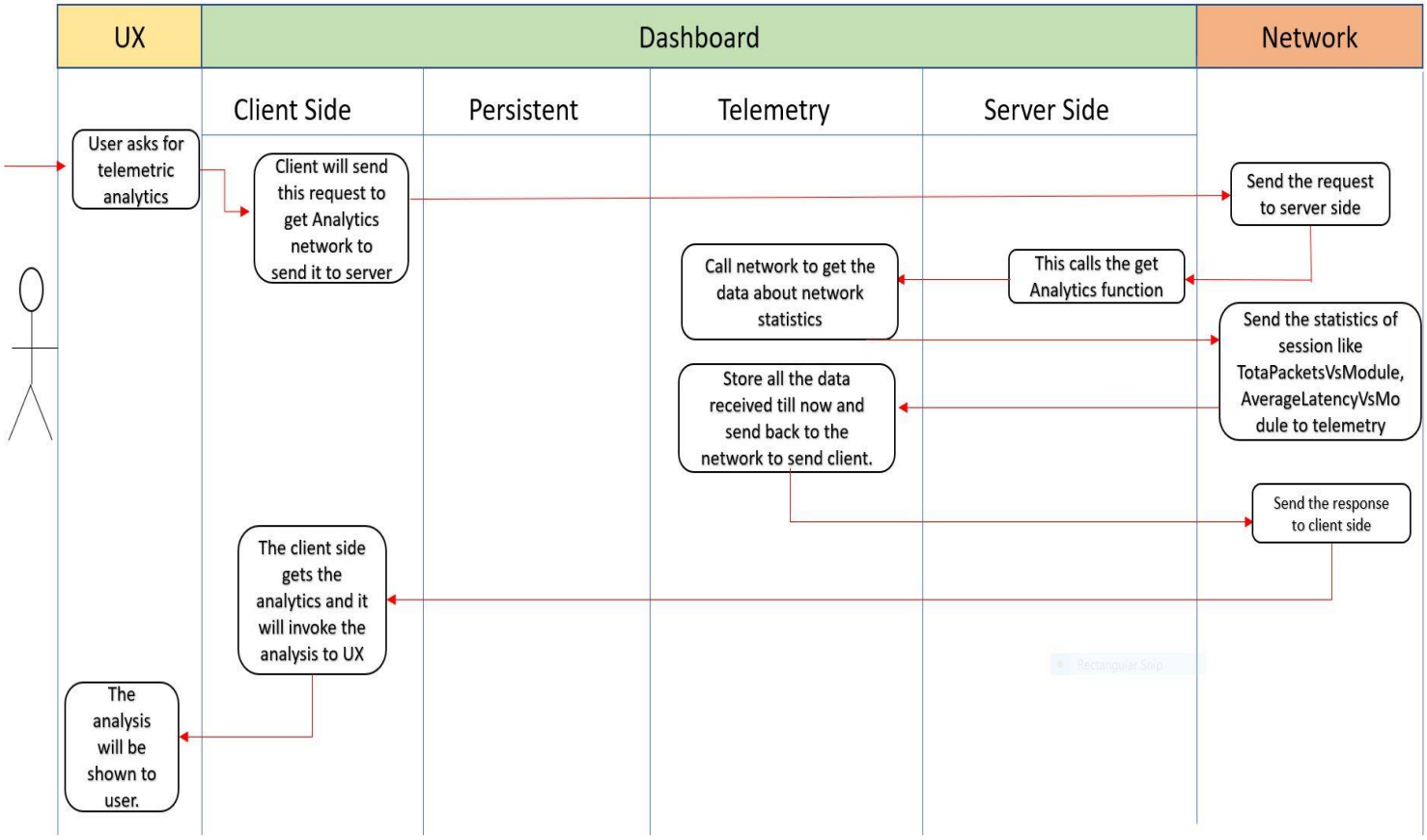
Lets see the activity diagram for this case.



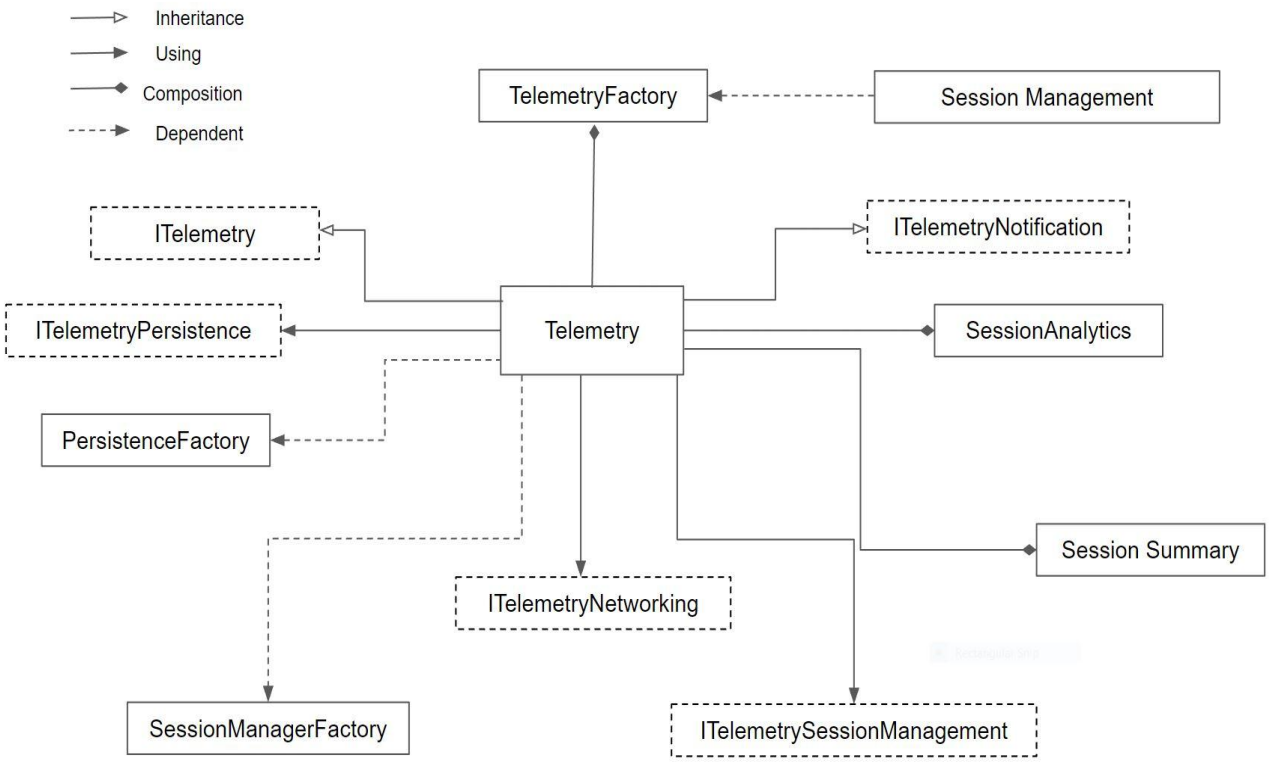
TELEMETRY DURING THE SESSION

The user can also choose to see the telemetry analytics during the session. In this case again UX will send the request to the client using **getAnalytics** and the client will send the request to the server via a networking module for getting analytics.
The server will check the event type and call the **getAnalytics** function and send back the result to the client side and then to UX.

Let's look at the activity diagram.



UML DIAGRAM



In the above class diagram we can see that the telemetry module is mainly dependent on the session management, persistent and network module. There are interfaces as well that will be required by the telemetry module and there are interfaces which the telemetry module has to implement so that other modules can use it.

DESIGN PATTERN USED

- **Factory design pattern** ⇒ in telemetry we have to implement the factory design pattern to make the abstraction for the client.
- **Single design pattern** ⇒ in TelemetryFactory we are using the singleton design pattern as we need only one instance of the telemetry object for the whole single session.
- **Publisher-Subscriber** ⇒ to get the notification about the change in session data the telemetry should be notified so that the telemetry can update the new telemetric data.

PERFORMANCE ANALYSIS

Fetching the data about the total number of chats per user, current user number is easy and does not have the performance overhead. But getting the data from the network module about the total number of packets that has been transmitted through the network module for every module has some overhead.

FACTORY

We will be using the factory pattern and singleton design pattern for telemetry.

```
// defining the factory for the telemetry module
public static class TelemetryFactory
{
    //defining the telemetry
    private static Telemetry telemetry = null;

    public static Telemetry GetTelemetryInstance()
    {
        // checking if telemetry instance already created
        if(telemetry == null)
        {
            telemetry = new Telemetry();
        }
        return telemetry;
    }
}
```

INTERFACES

Lets see the interfaces that telemetry module will be implementing and also interfaces that telemetry module requires to be implemented by other modules.

There will be two types of interfaces

1. Interfaces which i telemetry module will be implementing and will be available to other modules.
2. Interfaces which will telemetry need to get data and information from other modules as well.

Interfaces which telemetry module will implement

ITelemetry

This interface will be required by the session manager so that session manager can call the saveAnalytics and getAnalytics function to store and get the information about the telemetry analytics.

```
// defining the interface ITelemetry which will be used to save the analytics and get
the analytics
public interface ITelemetry
{
    //  the function to save the analytics once the meeting
    public void SaveAnalytics(<Data_Type> chatMessages);

    // function to get the analytics of the ongoing session to show the client
    public SessionAnalytics GetTelemetricAnalytics(<Data_Type> chatMessages);
}
```

Interfaces which telemetry will need from other modules.

ITelemetryNetworking ⇒ this interface will be implemented by the networking team. The telemetry module will be using this interface to get the details about the total number of packets for a given module that has been transmitted over the network. And also should provide the average latency.

```
// defining the interface for getting data from the network module
public interface ITelemetryNetworking
{
    // function to get the dictionary of total packets vs module
    public Dictionary<string, int> getTotalPacketsVsModule();

    // function to get the dictionary of average latency vs each module
    public Dictionary<string, float> getAverageLatencyVsModule();
}
```

ITelemetryNotification ⇒ this interface will be implemented by the session manager.

```
// interface to update the session and telemetry once there is a change in the session
public interface ITelemetryNotification
{
    // function handler whenever the current session updates.
    void OnSessionChangeUpdateAnalytics(SessionData newSession);
}
```

ITelemetrySessionManager ⇒ this interface provides the telemetry to be able to subscribe to be able to listen whenever the session data is changed.

```
// interface between the session manager and telemetry for subscribing to the session
energy.
// By subscribing to this the telemetry will be able to use the method of session
manager.
public interface ITelemetrySessionManager
{
    // function to subscribe the session manager to update the session data
    // the listener is the subscriber
    public void Subscribe(ITelemetryNotification listener);
}
```

ITelemetryPersistence ⇒ this interface will be implemented by the persistence submodule. The telemetry module will call the save function to save the telemetry data and the summary of the current session. The persistent will save it in the file.

```
// defining the interface for telemetry to save the telemetry anaytics using
persistence
public interface ITelemetryPersistence
{
    // function to save the analytics in the file after the session ends
    public void Save(SessionAnalytics sessionAnalytics);
}
```