A Design Specification Document for the course CS5617 (Software Engineering)

Design Specification Document

by

Neel Kabra

UX Team

(Neel Kabra)

(Parichita Das)

(Jasir K)

Under the supervision of

Ramaswamy Krishnan Chittur

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

Department of Computer Science and Engineering

Kerala, India - 678557

UI/UX Specs Document

Team Lead

Neel Kabra

Team Members

Jasir K

Parichita

Overview

The method design teams employ to produce products that offer customers meaningful and pertinent experiences is known as user experience (UX) design. The UX team for this project is responsible for joining all the UX components of all other teams. We have allocated certain resources (real estate) to all other teams for their UX components. We are responsible to ensure all components are displayed properly.

Required Features

1. Sign-in will be done through google to authenticate the user. This will also help the user avoid filling in some details like profile picture and name.
2. A home screen that will have options to create or join a meeting. There will also be a button which will redirect to cloud's UX page.
3. A splash screen that will display our logo.
4. Link between UX of the application and the UX for all the respective modules. Tabs will be used to change the UX and link it to the dashboard UX, whiteboard UX, and screenshare UX. A floating button will be used to access the UX of the chat. The chat window will be a pop-up.
5. A message will be displayed when someone leaves the meeting, regardless of what module's UX is being displayed.
6. There will be a toggleable light and dark mode. This can be toggled anywhere regardless of what module's UX is being displayed, from home screen to all the different UX screens.

7. Shortcut keys - Support to use shortcuts like `ctrl + tab` to move to the next tab, or `ctrl+shift+tab` to move to the previous tab will be added.
8. Notify on the main screen for when someone leaves a meeting.

Task Division

We have broken down the tasks of the UX team into the following

1. Splash Screen (Neel Kabra)

   The splash screen is the first screen people will see when the application opens. This will generally be just for half a second or so, and it will mostly feature our logo along with our name and a small quote or so.

2. Authentication (Parichita Das)

   We have decided to try to implement Google Sign-In for authentication. We will use the **OAuth2.0** framework to validate whether the client/server is authorized to join the meeting(from IITPKD).

3. Home Screen (Jasir K)

   After the authentication is done, we will redirect them to the Home Screen. This screen will have three options

   - Host a meeting
     - Makes current system as server
     - Provides a button to start meeting
     - Generates a sessionID and userID
     - Passes on the name, profile picture, userID and sessionID to the dashboard module.
   - Join a meeting - This option will take the server's IP address, and the sessionID (for validation), and pass on details to the Dashboard.
     - Takes name, roll number received from GoogleAuth
     - Takes the servers' IP address (maybe also their own if required)
     - Takes the sessionID
     - Provides a button to join the meeting
     - Send all the relevant details to the dashboard module.

- Retrieve session (for cloud) - There will be a button for viewing previous sessions, which will redirect to a new page containing all the session details which would be implemented by the Cloud team.

4. UI Layouts (Neel Kabra)

   We need to make sure the following UX components are linked - Dashboard, Whiteboard, Screen Share and Chat.

Design Patterns

MVC

- **Model** (**M**): This is the bottom most layer whose objects hold information of all the data. Most of the business logic is written here.
- **View** (**V**): In the MVC design pattern, the view listens to the model for any changes that are required to be displayed. The view also incorporates changes from the controller.
- **Controller** (**C**) : It is the Controller's job to handle incoming requests. It uses the Model to process the user's data before returning the outcomes to the View. Typically, it serves as a go-between for the View and the Model.

MVP

- **Model and view are similar to above**
- **Presenter** (**P**) : The presenter mainly takes input from the user through the view, sends the data to the model for processing, and then sends it back to the view to display. This leads to there being no direct communication between the view and the model, which is really helpful for unit testing.

MVVM

- **Model** (**M**): This is the bottom most layer whose objects hold information of all the data. This is also known as the backend, where all memory is managed and all our algorithms are written.
- **View** (**V**): This is what the user sees. This is primarily written in XAML.

- **ViewModel** (**VM**): This is responsible for binding the view to the model.

We will most likely be going for the MVVM design pattern for the UX.

Splash screen

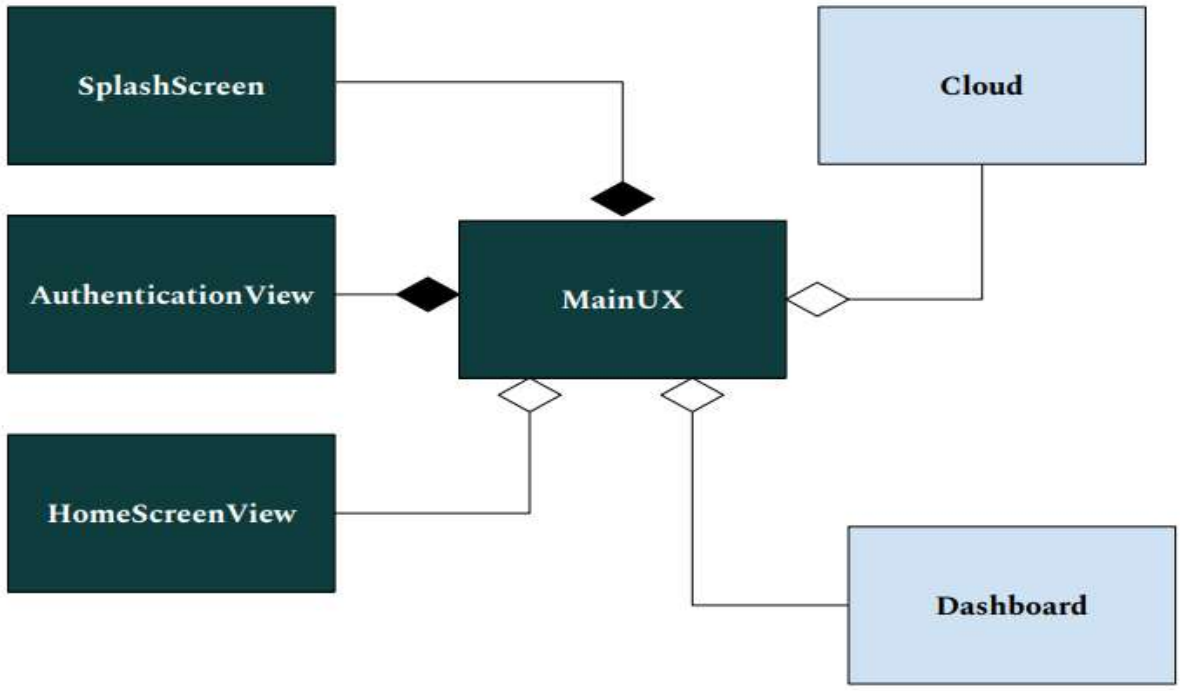
This is the screen that will open up when starting the application. This will simply be a view. We will incorporate an animation into the splash screen which will feature our logo and hopefully a good quote or one liner about what our application does.

We can easily implement this by showing a splash screen and calling thread.Sleep for some arbitrary time.

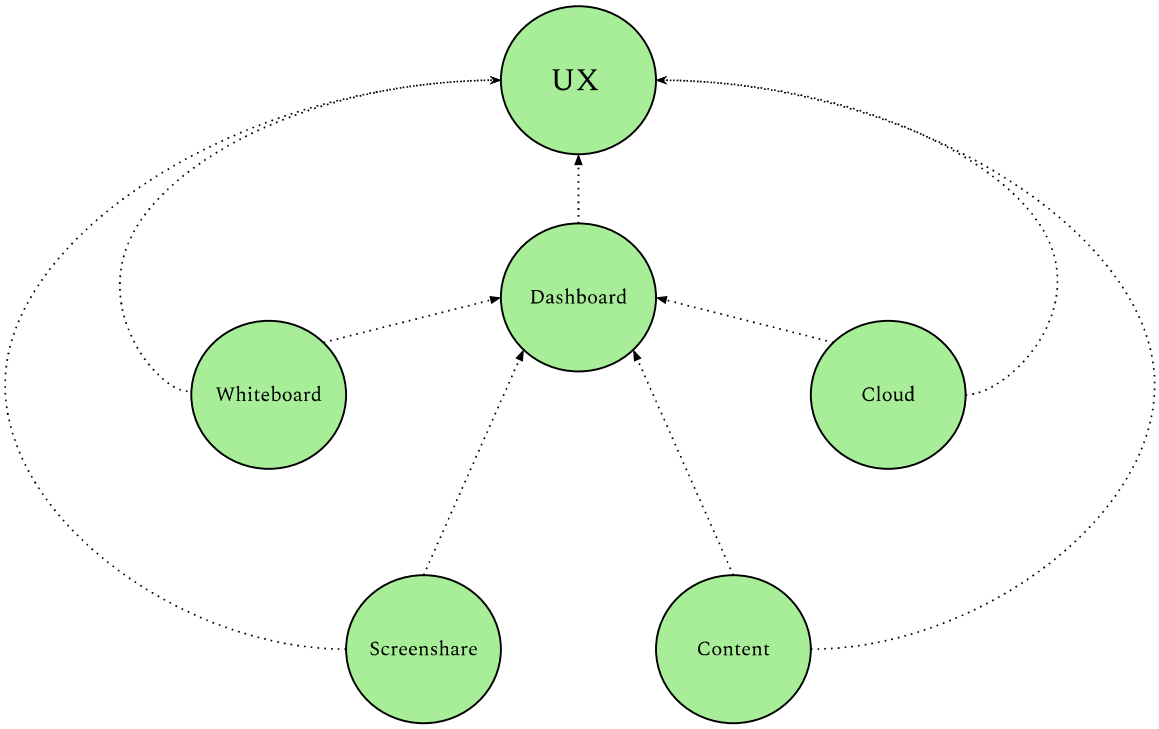
Code example :

```
SplashScreen splash = new SplashScreen();
splash.Show();
Thread.sleep(); // for some arbitrary time
splash.Close();
```

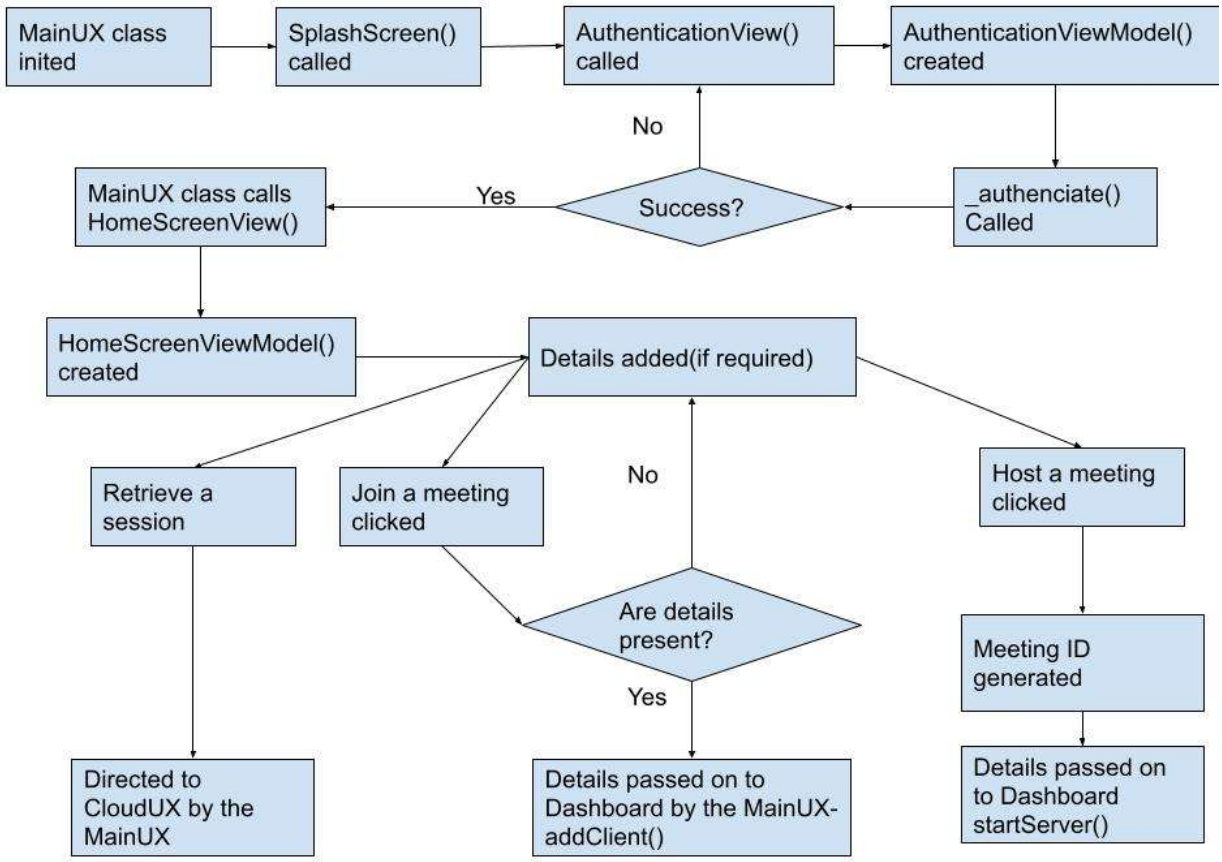
UML Diagrams



In the above diagram, it is clear that MainUX is composing the *SplashScreen* and *AuthenticationView* as both of them must be constructed if *MainUX* gets instantiated. Once Authentication is done, it returns the struct/object with User Information to the *MainUX*. It might happen that the user does not get authenticated and hence the *MainUX* will not proceed from here onwards. In the other case the MainUX will now create a *HomeScreenView* object and pass user information to it. The Home Screen is now in charge of what the next steps from here will be. In case of Join/Host a new meeting, the Home Screen module sends back respective information to the *MainUX* asking it to create a *Dashboard* object, otherwise in case of retrieving an old session, the *MainUX* is redirected to create a *Cloud* object. This way *MainUX* aggregates *HomeScreen*, *Dashboard*, and *Cloud* modules.



Module Diagram



Activity Diagram

UI Layouts - Dashboard, Screenshare, Whiteboard

We have decided to go with a tab layout for the UX.
Each Module will have around 90-95% space length-wise, and 100% of the space width-wise.

Clicking on the tabs will directly link to the views of the Dashboard, Whiteboard and Screen Share. The UX for all modules will be done by their team itself, so the UX team is responsible for just making sure the respective views are linked.

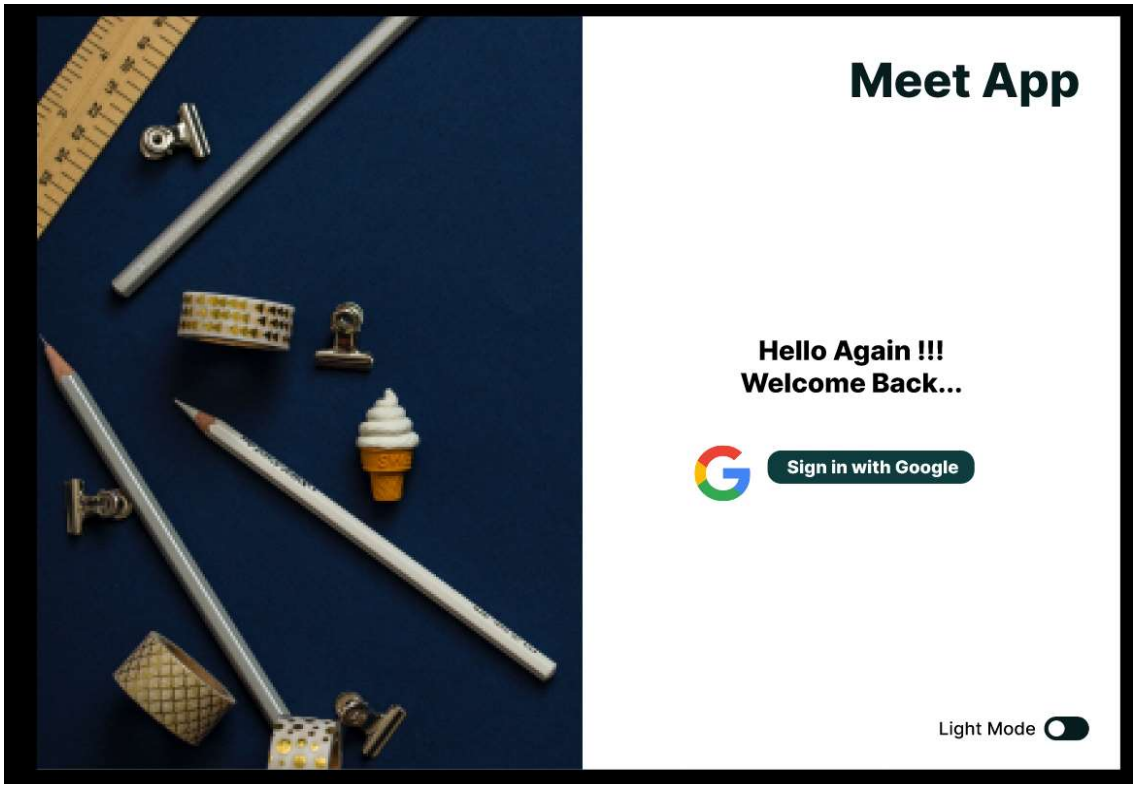
The teams will be made aware of the color schemes that are chosen.

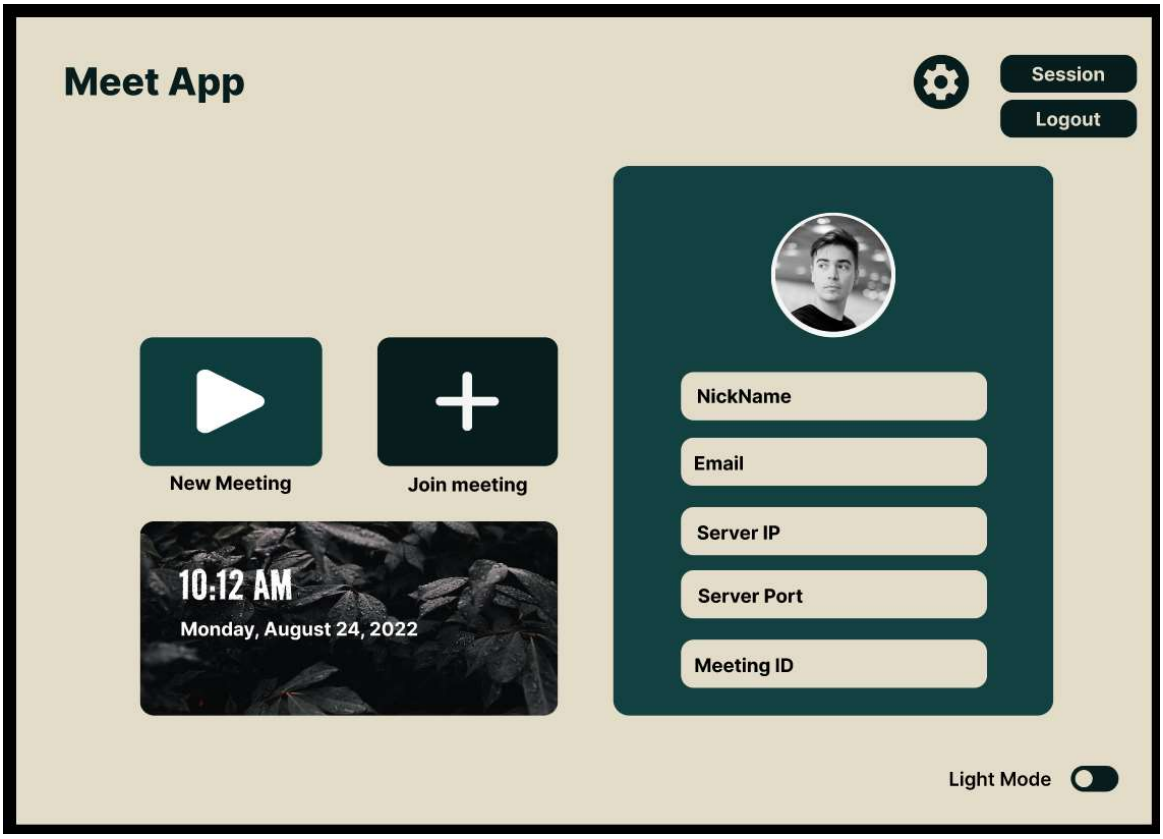
For chat, a pop feature will be implemented. The chat bubble will indicate that a new message has been received, and clicking on the chat bubble leads to the chat window opening up. The real estate has not been given yet, but some approximate figures have been given.

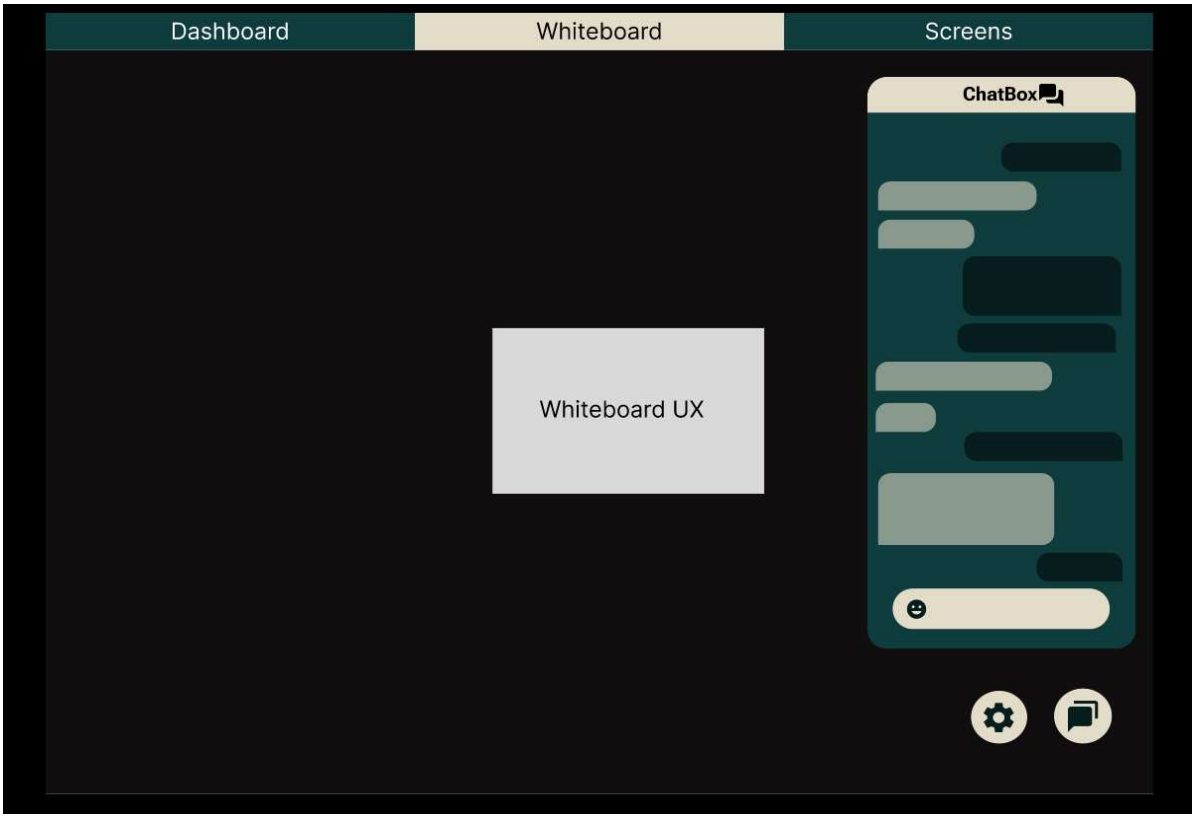
The link to the UX model is :
https://www.figma.com/proto/6OcWQH82ImzvPBnYOXRogk/Untitled?node-id=2%3A8&scaling=contain&page-id=0%3A1&starting-point-node-id=2%3A8

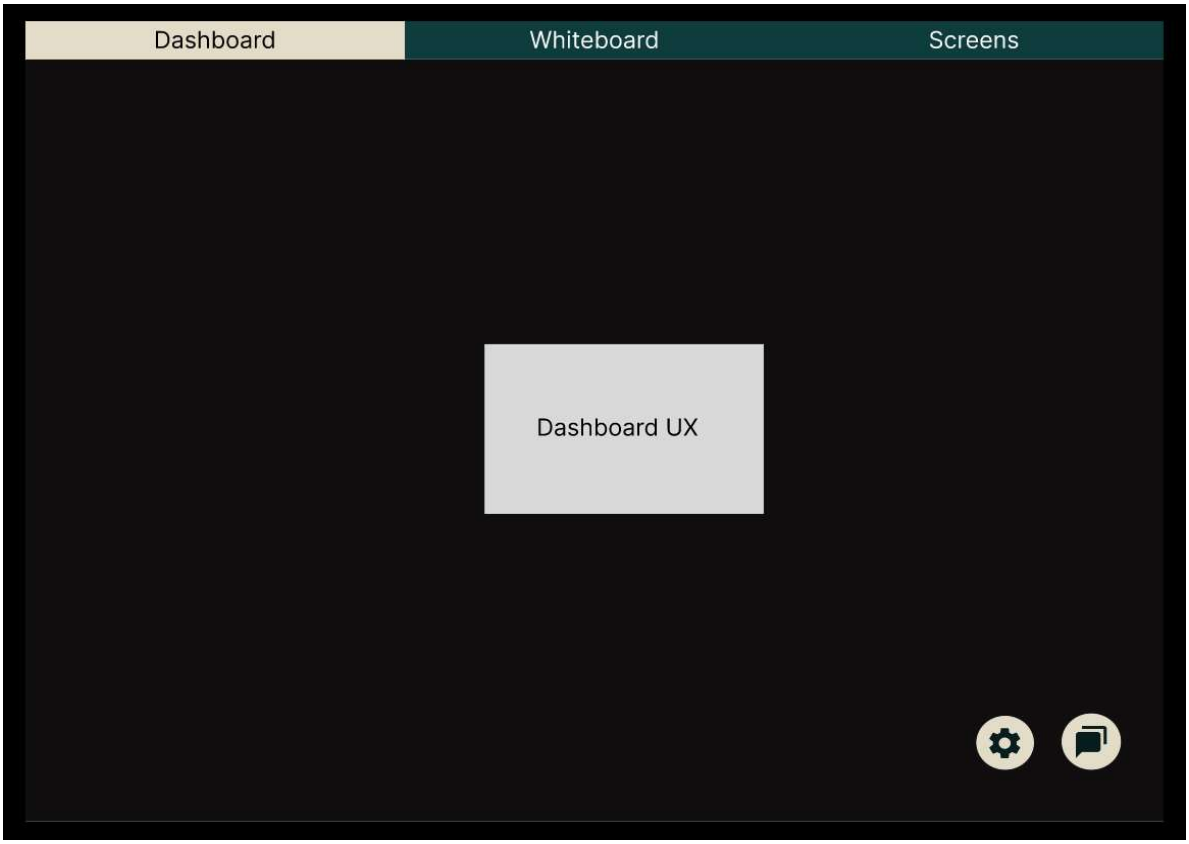
This model is clickable and can be played around with it.

Prototype images :




Meet App
Session
Logout
New Meeting
Join meeting
10:12 AM
Monday, August 24, 2022
NickName
Email
Server IP
Server Port
Meeting ID
Light Mode


Dashboard
Whiteboard
Screens
ChatBox
Whiteboard UX


Dashboard
Whiteboard
Screens
Dashboard UX

Code sample:

View (.xaml)

```
<container1>
            <controlbutton1 click = dashBoard_click text =
"Dashboard"></controlbutton1>
</container1>

<container2>
            <controlbutton2 click = whiteBoard_click text =
"Whiteboard"></controlbutton2>
</container2>

<container3>
            <controlbutton3 click = screenShare_click text =
"Screenshare"></controlbutton3>
</container3>

<container4>
            <controlbutton4 click = content_click text = "Chat">
</controlbutton4>
</container4>
```

View

```
private void content_click(object sender, RoutedEventArgs s){
      // the tab for the Content has been called
      openContentUX()
}
private void screenShare_click(object sender, RoutedEventArgs s){
      // the tab for the Screen Share has been called
      openScreenShareUX()
}
```

```
private void whiteBoard_click(object sender, RoutedEventArgs s){
      // the tab for the WhiteBoard has been called
      openWhiteBoardUX()
}
private void dashBoard_click(object sender, RoutedEventArgs s){
      // the tab for the DashBoard has been called
      openDashBoardUX()
}
```

Conclusion

The UX module is responsible for linking all the module's UX and also providing an overall theme to the application. The module is also responsible for authentication and is essentially the first point of contact for the application. The module lies on top of the Dashboard module with Screenshare, Whiteboard, Content and Dashboard module being dependent on it. The module will create an object of the dashboard module, and pass on information to the dashboard module which in turn will broadcast it for whoever requires it. The UX module will also be responsible for displaying everyone's views when the main application is running. The transition of one modules' UX to another team's module should be handled by the UX module, along with the allocation of the real estate for the application.