Papertown App

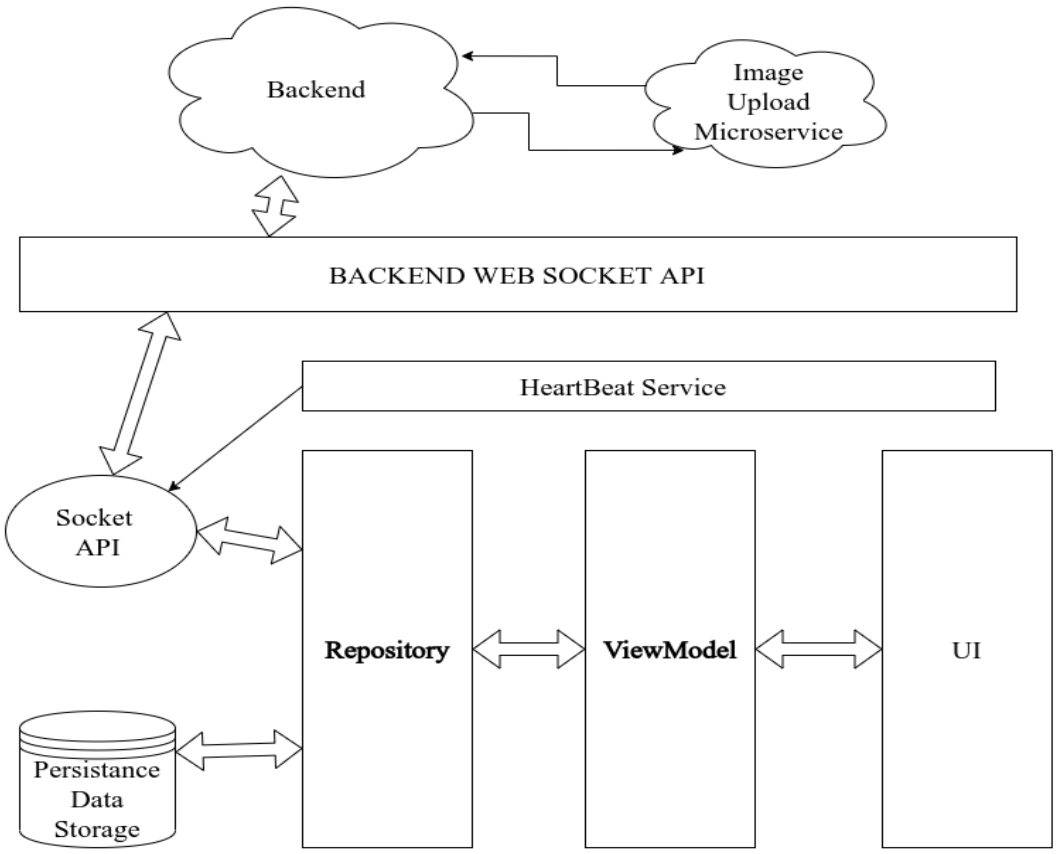
Papertown is a community building platform for creators to run their community easily and make revenue out of it. It makes it easy to share and sell their program. People can discuss the problems and learn together.
App link : https://play.google.com/store/apps/details?id=org.flinkhub.messenger2

This app has list of features:

1. Login/SignUp
2. Chat
3. Feed [like Twitter, Instagram, LinkedIn]
4. Search
5. Creating a new post

Here is the app's high level design.We are MVVM architecture here to make features independent or modular. Below the design for internal working of whole app :

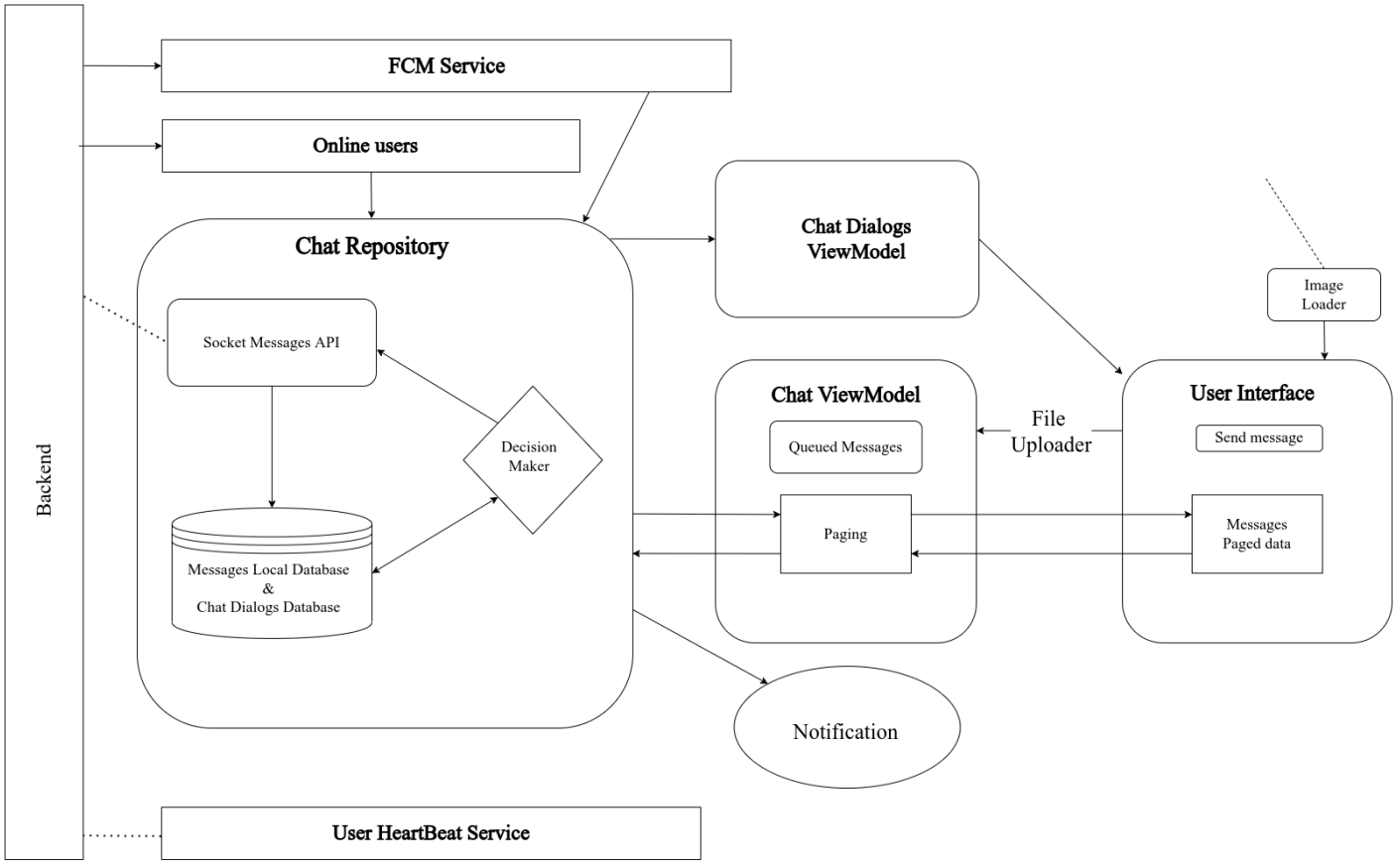
Diagram : High level app design



High Level App Design:

1. App is connected to a server using websockets. As soon as the user opens the app it gets connected to a web socket.
2. **UI :** It is the UI layer where the user is interacting with the app.
3. **ViewModel** : This layer's main purpose is to observe the data from the repository and update it on the UI. This makes sure that the UI always has updated data.
4. **Repository** : This layer is managing the network call and local database sync and serving the updated data to the viewmodel. It has **PageKeyedDataSource** which severs the pagedList to the viewmodel
5. **HeartBeat Service** : This is a service which is running in background and sending user online status every 5 seconds to the server. This service helps us in handling users last seen [For chat feature]. This service only runs when the app is in foreground and it can send events without login [It does not require user session for guest user analytics purposes].

Chat Feature Flow :



This feature helps creators and learners to communicate with each other

1. **UI :** At this layer user can send message and send attachments (like image, video, document, audio)
2. **Message PagedData** : This component gets a list of messages from the view model and as the user scrolls add new messages.
3. **Image Loader** : For this purpose we are using **Glide,** a third party library.
4. **File uploader** : This is a class which uploads files to our server, it is directly interacting with socket API and in response it gets *mediaObjectId* which is attached to the message and sent to the viewmodel. We are allowing limited file size for uploading that's why we are using this flow.
5. **ChatViewModel** : This layer continuously observes the changes in message data from the repository and actions from UI and refreshes itself accordingly. It also has queued messages data which it sends to the repository.
6. **ChatRepository** : This layer manages the data for messages , queued messages (in case of no internet connection) , userDialogs online and notifications. The decision maker is the one who is responsible for API calls required.
7. **Online users** : This is an event received from the server that has contacted the user's online status. This layer directly sends this to the repository which manages it further. It also gets notification when the user is running the app.
8. **FCM Service** :This is service from firebase which gets called when there is any notification when the app is in the background. We first clear that fcm default notification and store that notification in our database (using repository). Then creates notification based on its content.
9. **Notification** : It contains the chat ids and some metadata . On clicking it will open the respective user chat/ group chat.
10. **ChatDialog ViewModel** : This manages the list of chats and their online indicator. This list also have paging, it

Feed feature flow :

This is a user feed (similar to twitter feed). It contains the user posts, new towns, and new topics. It has a list of different types of items like image, text, audio, video etc. For video there is a support for auto play, it plays as the user scrolls and stops when that item goes behind the screen.

1. **UI** : This layer shows users a list of items with which users can interact.
2. **ExoPlayer** : We are using this library for media playback and media sessions. It will stream the videos from url.

3. **HomeFeed ViewModel** : This observes the changes in items and updates the UI accordingly. It interacts with the repository and gets new data.
4. **HomeFeed Repository** : This manages the list of items and checks for new posts and updates the local db. In response it will send the paging data to the viewmodel. We are using the ***Paging Library*** for paging the list of items. Specifically, we are using **PageKeyedDataSource** which returns the pagedList to the view model**.**