

# **ChatConnect – A Real-Time Chat and Communication App**

## **A PROJECT**

**Submitted by**

**M. INDHUMATHI (20201231506214)**

**P. PREMA (20201231506229)**

**A. SELVA VIGNESHWARI (20201231506235)**

**A. SUPRIYA (20201231506238)**

**MENTOR**

**Dr.T. ARUL RAJ M.Sc., M.Phil., Ph.D**



**DEPARTMENT OF COMPUTER SCIENCE**

**SRI PARAMAKALYANI COLLEGE**

**ALWARKURICHI – 627 412**

**APRIL - 2023**

## TABLE OF CONTENTS

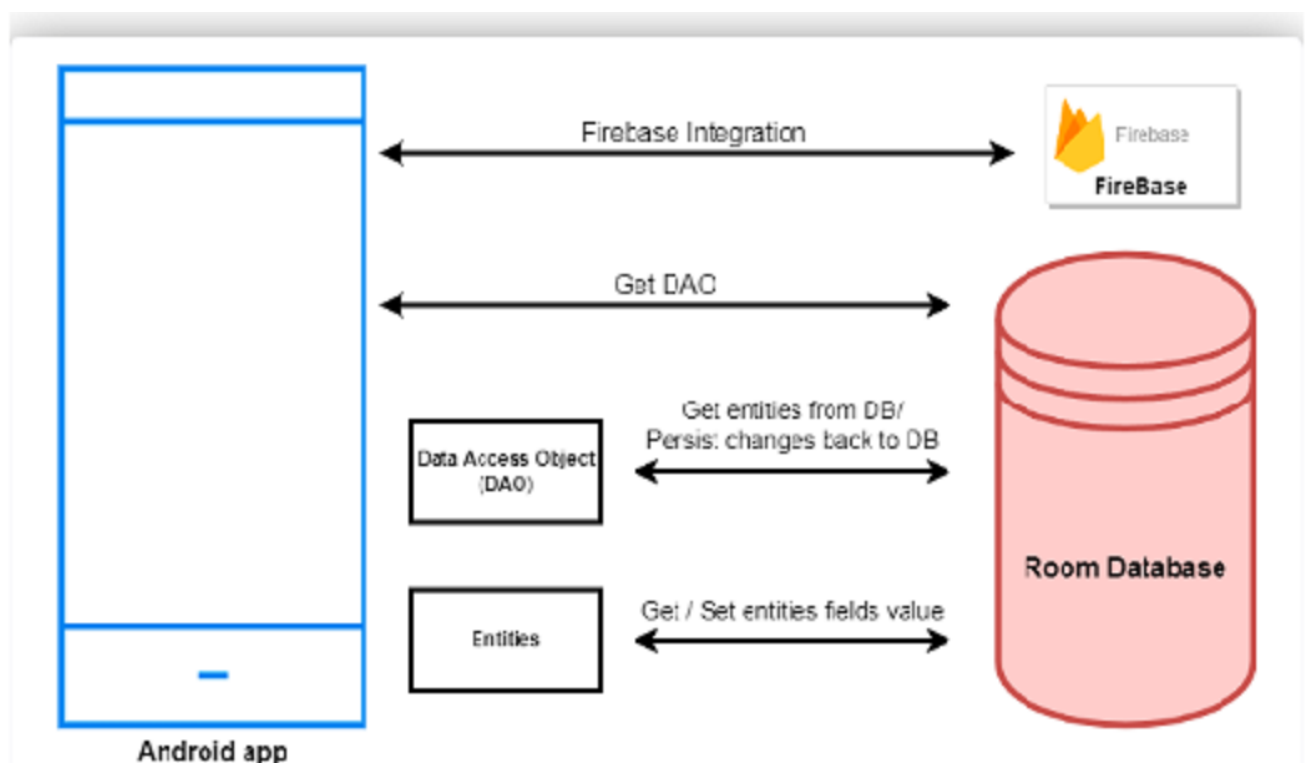
CHAPTER NO.	TITLE	PAGE NO.
<b>I</b>	<b>INTRODUCTION</b>	1
	1.1 Overview 1.2 Purpose	
<b>II</b>	<b>PROBLEM DEFINITION AND DESIGN THINKING</b>	6
	2.1 Empathy Map 2.2 Ideation and Brainstorming Map	
<b>III</b>	<b>RESULT</b>	11
<b>IV</b>	<b>ADVANTAGES AND DISADVANTAGES</b>	14
<b>V</b>	<b>APPLICATIONS</b>	18
<b>VI</b>	<b>CONCLUSION</b>	21
<b>VII</b>	<b>FUTURE SCOPE</b>	22
<b>VIII</b>	<b>APPENDIX</b>	23
	(i)Source Code (ii) Screen Shot	

# CHAPTER – I

## INTRODUCTION

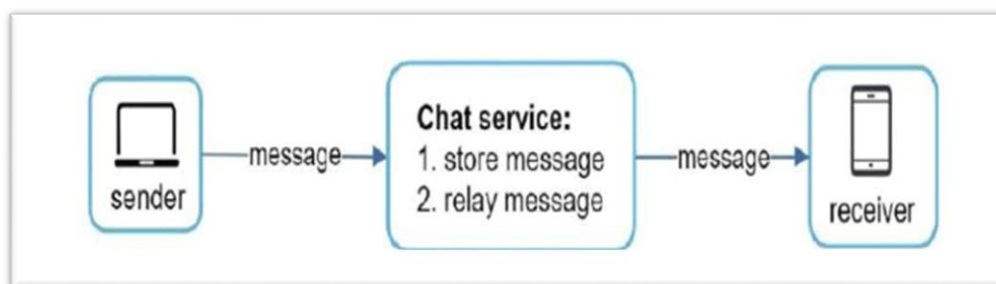
### 1.1 OVERVIEW

ChatConnect is a social networking tool that leverages on technology advancement thereby allowing its users communicate and share information. It offers a wonderful one stop shop experience for keeping in touch with people you know. It can be used for messaging and placing voice messages. Chatting app allows you to communicate with your customers in chat. It enables you to send and receive messages. Chatting apps make it easier, simpler, and faster to connect with everyone and it is also easy to use. ChatConnect is a sample project built using the Android Compose UI toolkit. It demonstrates how to create a simple chat app using the compose libraries. The app allows users to send and receive text messages. The project showcases the use of compose's declarative UI and state management capabilities. It also includes examples of how to handle input and navigation using composable functions and how to use data from a firebase to populate the UI.



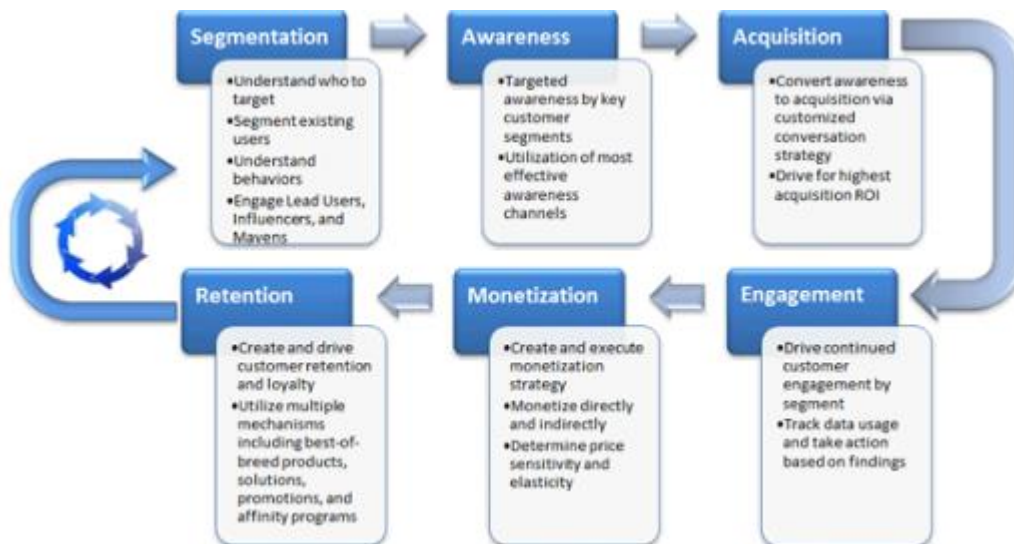
## 1.2PURPOSE

- The purpose of the Chat Connect is to allow users be able to chat with each other, like a normal chat application.
- The users will be able to chat with each other, most likely only from user to user, no group chatting will be developed, unless there is time to do so.
- A chat connect app makes it easy to communicate with people anywhere in the world by sending and receiving messages in real time.
- Also a messaging app, a ChatConnect is a platform that enables a user to connect and message another person through the use of their android mobile device.
- The project should be very easy to use enabling even a novice person to use it.
- A react real- time chat where users can chat anonymously with others and also can chat with others privately and get unread private messages from others.
- This type of chat allows for quick communication from the client who has a problem and the chance to get immediate answers and solutions.
- A reliable and secure web chat plays a large role in ensuring a positive user experience in app.
- That the messages between active users are being sent and received in real – time fosters an engaging chat where users can immediately interact with each other.



- If the client intends to start a conversation, it connects to the chat service using one or more network protocols. For chat service, network protocol selection is important.

- ChatConnect apps have made communication more interesting than any other regular chat service. They make us feel more connected to others, both personally and professionally on a daily basis.
- For example, chat apps like WhatsApp or messenger can deliver a meme or message from a friend that can truly make our day.
- With multi-tasking mechanism playing the major focus, today's chat apps are explored globally by billions of users for both personal as well as commercial fulfillment.
- At the heart of these Chat app innovation lies fascination for mobile technology that was earlier seen as a fleeting fad.
- The reason why chat apps are exercised with great sincerity is the fact that there is a subtle opportunity to transform chat application services into a major marketing and purchase channels. Artificial intelligence must also play a curious role here.
- As we progress in this blog, we are going to glance at how chat apps are playing several roles, including enhancing engagements, monetization and user retention and more.
- A lot of traffic and referrals observed on news organizations belong to social platforms, which translates into a great opportunity for relationship building and audience engagement.
- Chat messengers present themselves as representatives of socials on mobile and drive the size of traffic too huge to ignore.
- Chat app created an ecosystem that demonstrates immense uniqueness, each offering opportunity to engage with friends and connection in an unforeseen chat environment.
- Providing users with rich features, these apps deliver incredible live chat experience, allowing users to share text messages and emoticons.
- The ability to send numerous formats of media files among many friends as well as capability to create personalized groups make the experience even more exciting.
- As the chat apps surpassed the initial purpose of 'staying connected' by enabling style and elegance, the user engagement as a result grew more prolific than ever.



- There are chat apps that support you well if you wish to drive traffic back to your website or commercial mobile app. Depending on product and services, you can tempt and engage users on specific chat apps and tell them to click URL links shared in there.
- The chat apps with programmable robots can converse with users virtually or prepare readymade expressions and responses to minimize manual hassle.
- With the inception of chat platform that demonstrate diverse opportunities in the chat ecosystem, users can relish chat apps that create special moments for them as they connect, collaborate and share.
- For readers and viewers, this is a great way to come together and discuss a range of topics in a unique way. For instance, Chat app is one of the platform that let you create standalone chats where users can connect to discuss interesting topics.
- It gives them the convenience to connect without the need to spend a fortune and time to meet people in-person.
- Chat app are typically hosted on a server with an internet connection, enabling members from around the world to hold conversations about various topics.
- A chat app service is an application, software, or website-based service that customer service representatives use to communicate with consumers.

- ChatConnect app is an instantaneous form of communication. In fact, text messaging has become the primary interaction channel for millennials across the globe.
- When develop chat within an app, get to connect from 2 users with each other to a number of users within the same space. The latter especially works perfect for collaboration tools used by large enterprises.
- The user engagement is very high in a live chatting app as the response rate is quicker.

### **In Business Environment**

- Chat app allows employees to engage with one another and work together. Teams of workers can share ideas and solve company issues with one click.
- In time where people sit at computers all day makes sense of conversations to happen there. A chat app is also more efficient for communication versus email. And the main goal of a chat app is to collaborate.
- This means that its integration can lead to better employee performance and productivity.
- In professional front, Chat app helps us to instantly ping a colleague on an ongoing task.
- In a company, a team leader can inform about the meeting to their team members with there is no need to meet in front, they can communicate through this chat app.
- In Work from home method is, so useful to communicate with each other. They discuss about their project status with this they feel more connected to work together in the same project.

### **In Educational Environment**

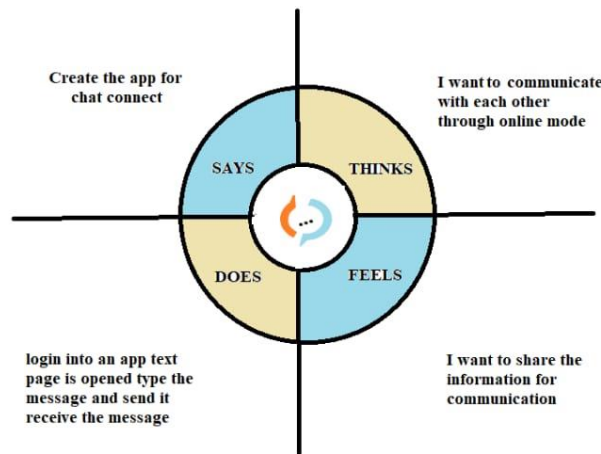
- Teachers can communicate with students to let them know about the yesterday or today class information for the absentees.
- Students can easily communicate to teachers if they have any doubt in subjects. It is mostly helpful during exam days.
- And also useful to share notes to the students within the students one another or the teacher can send to their students.

# CHAPTER – II

## PROBLEM DEFINITIONS AND DESIGN THINKING

### 2.1 Empathy Map

An empathy map is a widely used visualization tool within the field of UX and HCI practice. In relation to empathetic design, the primary purpose of an empathy map is to bridge the understanding of the end user. Within context of its application, this tool is used to build a shared understanding of the user's needs and provide context to a user - centered solution.



The traditional empathy map begins with four categories: says, thinks, does and feels.

#### Says category:

Creating the app for chatting with this people can communicate with one another easily and effectively. They can share their thoughts and feelings through this app.

#### Thinks category

Communicate each other through online mode to share information. Through sharing information people let know about the others and the things which is done by others.



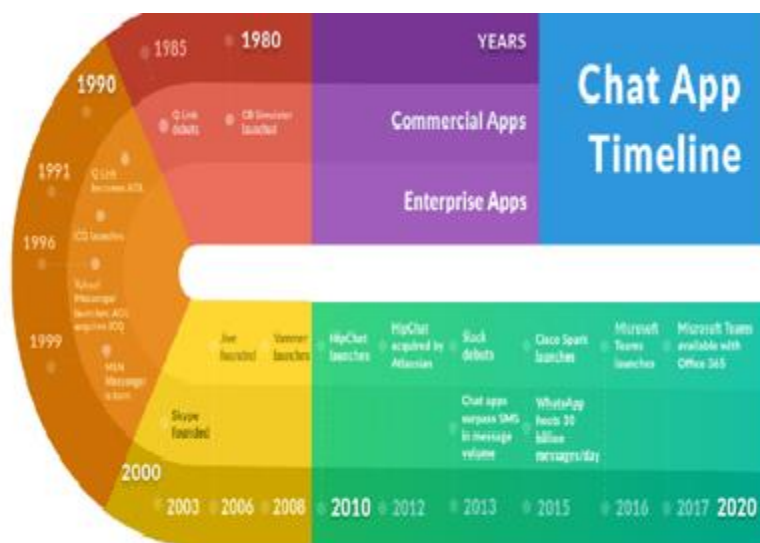
## Does category

User can register into an app and after registering the user can login into an app. After that the home page is opened with this the user can text the information what they want to share by one click they can share the information.

## Feels category

Sharing the information for communication to let them know each other, information like their thoughts, emotions, studies related, business related , information depend upon what they want to share with the other person.

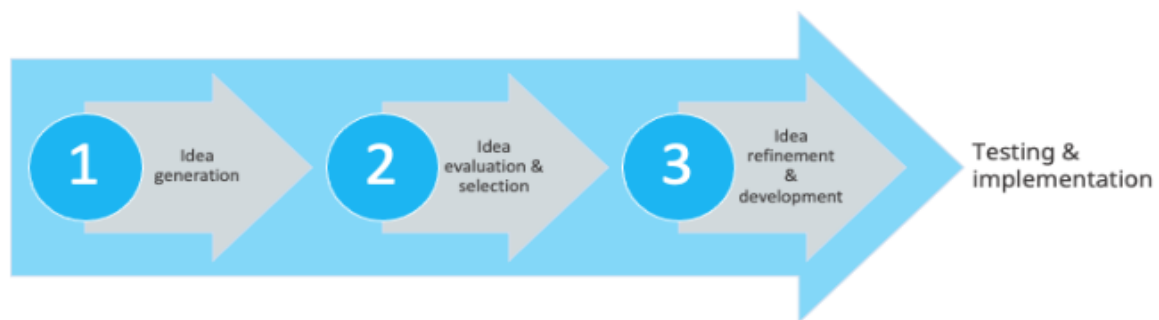
It is for the normal chat apps.



## 2.2 IDEATION AND BRAINSTORMING MAP

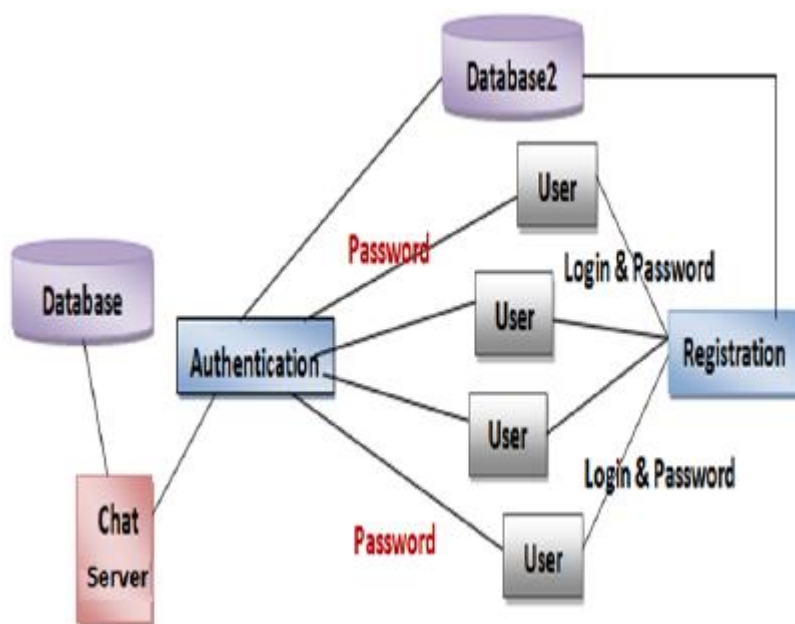
### Ideation

Ideation is the creative process of generating, developing, and communicating ideas. Ideation tools make the process organized to translate ideas in better form. The visuals and features of ideation tools bring clarity to the bumbled-up ideas. Thus, accelerating the further steps in the design thinking process. We can divide ideation in these three stages: generation, selection, and development.



### Brainstorming

Brainstorming is an activity that will help you generate more innovative ideas. It's one of many methods of ideation—the process of coming up with new ideas—and it's core to the design thinking process.



Physical Evidence	<div>Install app</div> <div>Viewing home page</div>
User Actions	<div>Register/log in</div> <div>Text page is opened and can type message to send</div>
Onstage Process	<div>The typed message was sent by one click</div> <div>Receiving message</div>
Backstage Process	<div>In key pad view of related texts which is previously typed</div> <div>Send messages through connection</div>

**Physical evidence:**

Installation of app in an android device. It is only for the android device.

**User Actions:**

In this app first of all, the user register to it after registration it view the text page to type the message. Once the user is registered in an app after the user can login to it and it shows the text page to send or receive a message.

**Onstage Process:**

The user can type the message and send it by one click. While another user is replied to the message the first user gets the reply. The sending and receiving message is visible in the same screen.

**Backstage Process:**

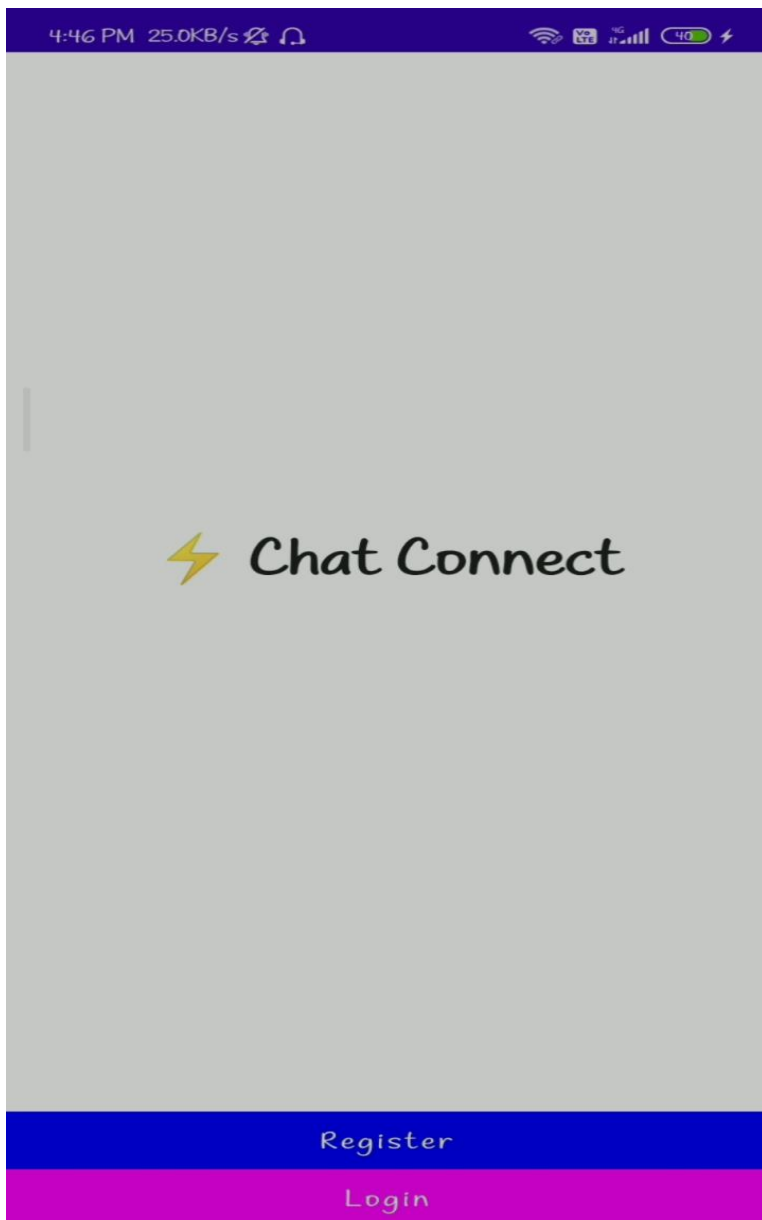
Over the connection the message can send from one android device to another android device. By this process the users can send and receive messages.

The chat system follows the client server model which provides reliable delivery of messages between clients. They can communicate with each other through this ChatConnect app.

## CHAPTER – III

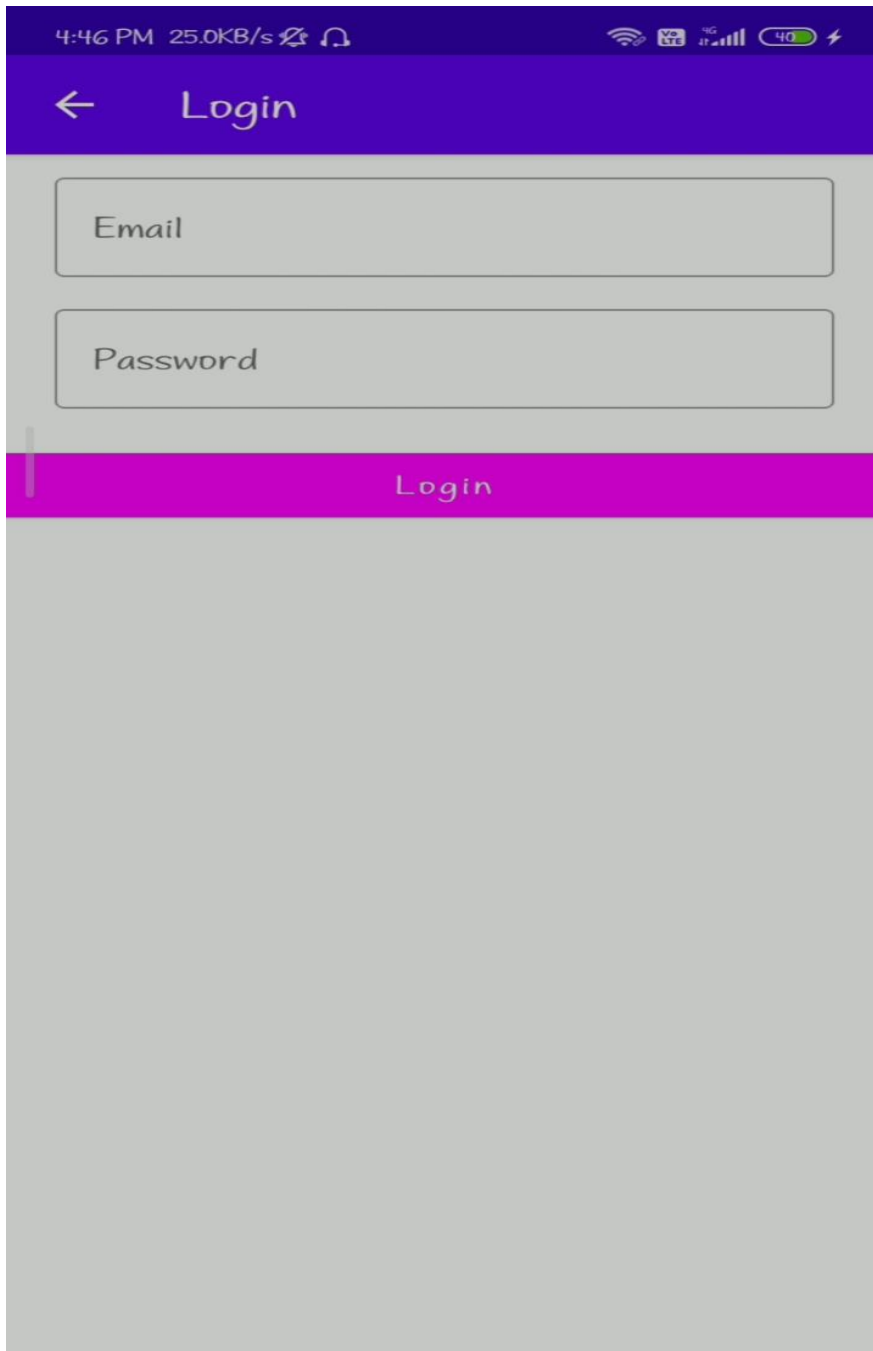
### RESULT

#### Logo Page



# LOGIN PAGE

AFTER REGISTRATION USER LOGIN INTO APPLICATION



The image shows a mobile application interface for a login page. At the top, there is a status bar with the time 4:46 PM, a download speed of 25.0KB/s, and various system icons including Wi-Fi, cellular signal, and battery. Below the status bar is a purple header bar with a white back arrow icon and the text "Login". The main content area has a light gray background and contains two white input fields with rounded corners. The first field is labeled "Email" and the second field is labeled "Password". Below these fields is a magenta bar with the text "Login" in white. The bottom half of the screen is a solid light gray area.

4:46 PM 25.0KB/s

← Login

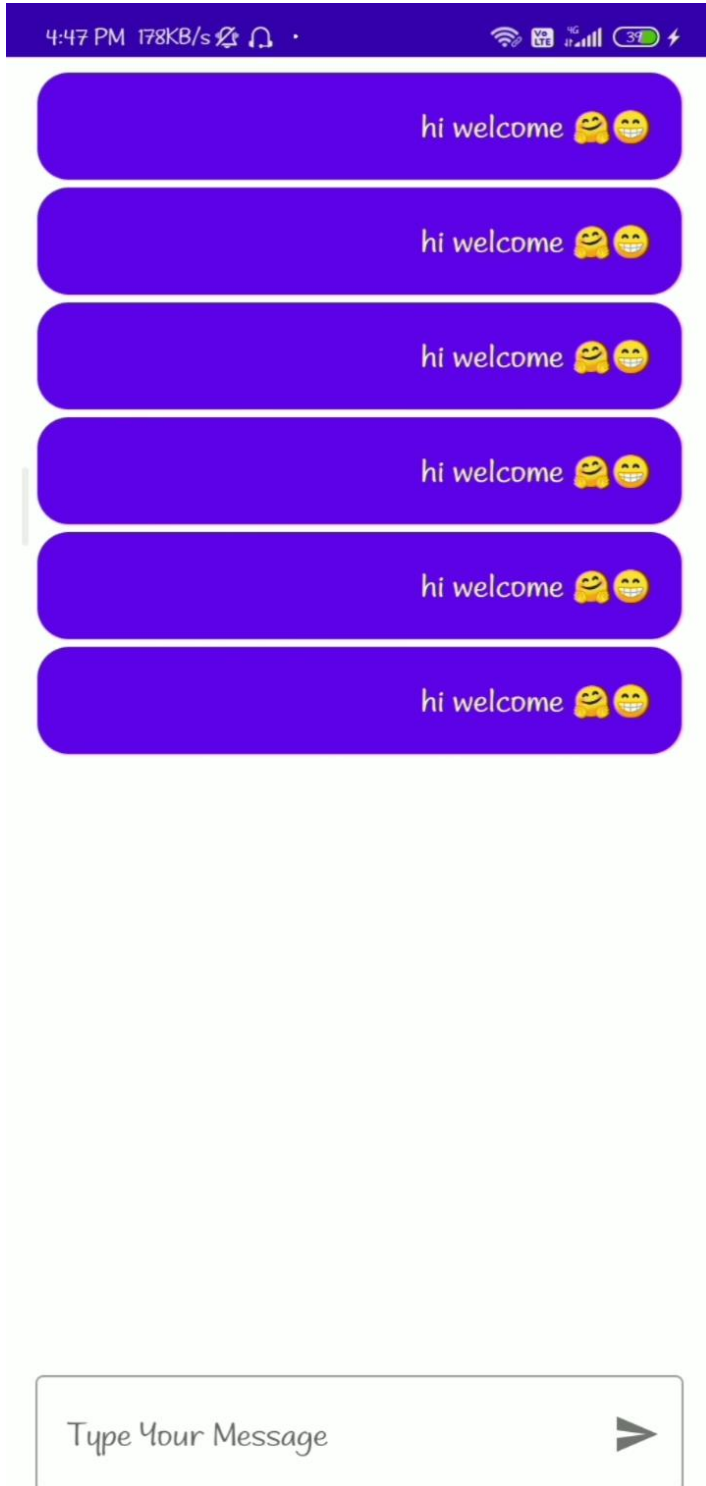
Email

Password

Login

# MAIN PAGE

AFTER LOGIN USER ENTER INTO THIS PAGE



## **CHAPTER – IV**

### **ADVANTAGES AND DISADVANTAGES**

#### **4.1 Advantages**

- Speed. A chat application allows you to message or contact a person in real-time.
- Familiarity.
- Convenience.
- Segmented Target Advertising.
- Increased Productivity.
- Employee Engagement.
- Privacy.

#### **Instant messaging advantages**

There are advantages and disadvantages of instant messaging. Some benefits of instant messaging include:

##### **1. Expediency of communication**

Because instant messages are just that – instant messages – they can help facilitate short, sharp conversations and allow people to share information quickly.

##### **2. More visible than email**

Instant messenger systems are more likely to be paid attention to by employees than emails are. Emails can sit in peoples' inboxes for hours unnoticed – if they are even opened at all – while the nature of instant messages is that they capture peoples' attention more quickly.



### **3. Easily connects people regardless of location**

Whether you're on the other side of the office, or the other side of the world, an instant messenger will let you contact a colleague and share ideas and information quickly, in real time. It's especially useful in situations where colleagues are working in remote or hybrid flexible styles.

### **4. Easy to use**

Instant messengers are intuitive and easy to use, requiring very minimal technical skill, making them a democratic communications tool in any workplace. People are used to using them in their daily lives to talk to family and friends and business instant messaging tools are a natural extension of this.

## **4.2 Disadvantages**

### **Instant messaging**

There are several good reasons to use instant messenger services in your organization: they enable employees to quickly reach out to one another to get information they need in a less disruptive manner than making a phone call, sending an email or wandering over to a colleague's desk for a discussion. Whether the person is at the next desk or on the other side of the world, this can be a very efficient form of informal communication, to this extent, and encourage collaboration and boost your team building.

However, it should be noted that instant messenger systems have numerous limitations and should not be relied upon as a major communications channel within your organization for imparting important communications or even for sharing sensitive information.

While each instant messaging system has its own pros and cons and configurations, in general these are the disadvantages your company can face when using instant messaging for business communications:

#### **1.Productivity issues with instant messengers**

While instant messaging for business can cut down on a lot of unnecessary time spent on ineffective communication, and reduce long-distance telephone charges, they are also potentially open to misuse.

Some cohorts of employees could spend hours each day using instant messenger, gossiping or generally wasting time while looking to the world like they are actually busy, doing work.

Messenger notifications can also still be just as distracting as a phone call or incoming email for someone who is trying to concentrate on some critical work or attempting to meet a deadline.

## **2. Security issues with instant messengers**

Very simply, most instant messenger platforms aren't secure and you should not trust them to send information that you wouldn't want being shared outside of your company.

The external instant messenger platforms are essentially on public networks, which means that they could easily be intercepted or hacked by external parties. Some instant messengers have better security protocols than others.

Your organization goes to a lot of effort to protect its classified, confidential or otherwise secret business files and store them securely on servers that can't be easily accessed by the outside world. When files are shared over instant messenger, all these security protocols go out the window unless you know you can trust that it is properly encrypted.

Chats conducted over an instant messenger can also be easily copied and pasted and end up in the wrong hands.

There are other vulnerabilities too around passwords and authentication, meaning information can fall into the wrong hands.

## **3. Ineffective tool for mass communication**

As mentioned above, instant messaging for business is a good place for informal chat and for quick exchanges of information between two people or very small groups. Beyond that, they become unwieldy and it can be hard for everyone to keep up with the chat when many, many voices are added to the mix.

So if you wanted to use one to broadcast information to all employees as a way of grabbing their attention, you are more likely to have your message drowned out by other voices chiming in... or it might go unnoticed entirely.

While instant messaging systems are a good way to bypass email – which has become a problematic way to share information in and of itself with many office workers suffering from email fatigue and not opening every message that hits their inboxes – it can also suffer from some of the same setbacks.

Instant messages cut down on small messages that can contribute to the volumes of electronic mail cluttering peoples' inboxes. But many workers see them as another annoying distraction and switch them off or mute them... or never even log into them in the first place. Many workers see it as “just one more thing” they have to do, and another thing that becomes distracting and overwhelming. These workers will never be reached if you have something urgent to impart.

#### **4. Problematic system for archiving**

Some instant messenger systems won't enable you to search for previous conversations when you log in to a new session, meaning you can't go back and look for previously shared information at a future date.

Other instant messenger systems have archiving functions, but you would need to develop a whole range of protocols around what to name the files so they can be useful and searchable.

A primary reason to improve internal communication within an organization is to facilitate information sharing and to improve corporate knowledge, and break down “silos”. Many instant messenger programs will not help with this, and only contribute to it.

#### **5. Cannot be controlled by management**

While certain features can be turned on and off, depending on the system, ultimately users of corporate instant messaging can send their own content to one another and it might not be correct, useful or constructive. In fact, it could even be contradictory to official information that management is sending elsewhere.

# CHAPTER – V

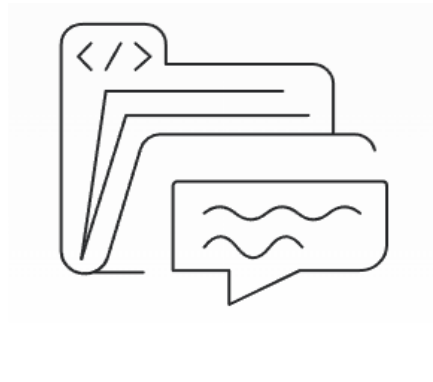
## APPLICATIONS

Quickly integrate a reliable & full featured chat experience into any mobile or web app. Our platform is flexible and designed to help you ship in-app chat quickly.

### Easy to Use

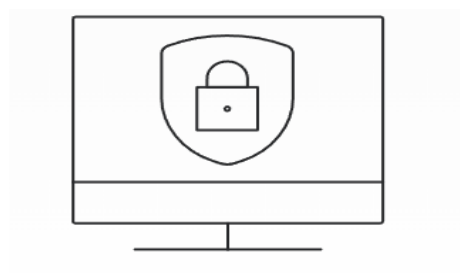
#### 1.Install App

They're available for Android. And they all work together, so cross-platform is easily set up.



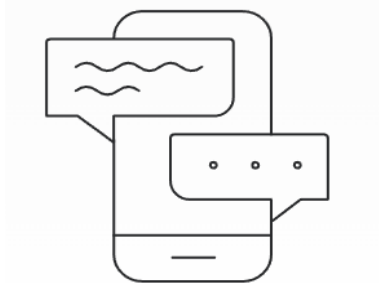
#### 2. Connect Securely

A single line of code sets up a secure connection to our service using the same underlying protocol as WhatsApp.



### 3. Build the Experience

Use your UI elements with our SDKs and build the features & extensions you need to create the complete experience.



### Build Confidently & Ship Quickly

Our idiomatic app are designed to help you build quickly. Available for Android, they all work together, making cross-platform a breeze. With documentation that's best in class, in-depth tutorials, and demo apps for every platform, you'll swiftly figure out exactly how they work.

### A Solution for Every Use Case

#### Social Community

Allow like-minded people in your online community and keep the engagement on your platform instead of relying to use external solutions.



## Marketplace

Deliver convenience and boost transactions by enabling communications between the buyers and sellers in your marketplace.



## On Demand

Delight your users by letting them get in touch with drivers, couriers, handymen, and other service providers all within your app.



## Edu –Tech

Connect mentors with mentees, teachers with their class, or even teachers with teachers.



## **CHAPTER – VI**

### **CONCLUSION**

The chat app provides a better and more flexible chat system. Developed with the latest technology in the way of providing a reliable system. The main advantage of the system is instant messaging, real – world communication, added security etc. This application may find the demand for most organizations that aim to have independent applications.

## **CHAPTER – VII**

### **FUTURE SCOPE**

With the knowledge I have gained by developing this application, I am confident that in the future I can make the application more effectively by adding this services.

- Extending this application by providing Authorisation service.
- Creating Database and maintaining users.
- Increasing the effectiveness of the application by providing group chats.
- Increasing the effectiveness of the application by providing Voice Chat.



## CHAPTER – VIII

### APPENDIX

**SOURCE CODE:** <https://github.com/Indhumj/chatconnect>

#### **MainActivity.kt file**

```
package com.project.pradyotprakash.flashchat

import android.os.Bundle

import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent

import com.google.firebase.FirebaseApp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        FirebaseApp.initializeApp(this)

        setContent {

            NavComposeApp()

        }

    }

}
```

```
}
```

## **Creating NavComposeApp.kt file**

```
package com.project.pradyotprakash.flashchat

import androidx.compose.runtime.Composable

import androidx.compose.runtime.remember

import androidx.navigation.compose.NavHost

import androidx.navigation.compose.composable

import androidx.navigation.compose.rememberNavController

import com.google.firebase.auth.FirebaseAuth

import com.project.pradyotprakash.flashchat.nav.Action

import com.project.pradyotprakash.flashchat.nav.Destination.AuthenticationOption

import com.project.pradyotprakash.flashchat.nav.Destination.Home

import com.project.pradyotprakash.flashchat.nav.Destination.Login

import com.project.pradyotprakash.flashchat.nav.Destination.Register

import com.project.pradyotprakash.flashchat.ui.theme.FlashChatTheme

import com.project.pradyotprakash.flashchat.view.AuthenticationView

import com.project.pradyotprakash.flashchat.view.home.HomeView

import com.project.pradyotprakash.flashchat.view.login.LoginView

import com.project.pradyotprakash.flashchat.view.register.RegisterView
```

@Composable

```
fun NavComposeApp() {
```

```
    val navController = rememberNavController()
```

```
    val actions = remember(navController) { Action(navController) }
```

```
    FlashChatTheme {
```

```
        NavHost(
```

```
            navController = navController,
```

```
            startDestination =
```

```
            if (FirebaseAuth.getInstance().currentUser != null)
```

```
                Home
```

```
            else
```

```
                AuthenticationOption
```

```
        ) {
```

```
            composable(AuthenticationOption) {
```

```
                AuthenticationView(
```

```
                    register = actions.register,
```

```
                    login = actions.login
```

```
                )
```

```
            }
```

```
            composable(Register) {
```

```
                RegisterView(
```

```

        home = actions.home,

        back = actions.navigateBack

    )

}

composable(Login) {

    LoginView(

        home = actions.home,

        back = actions.navigateBack

    )

}

composable(Home) {

    HomeView()

}

}

}

}

```

## Creating Constants object

```
package com.project.pradyotprakash.flashchat
```

```
object Constants {
```

```
    const val TAG = "flash-chat"
```

```
const val MESSAGES = "messages"

const val MESSAGE = "message"

const val SENT_BY = "sent_by"

const val SENT_ON = "sent_on"

const val IS_CURRENT_USER = "is_current_user"

}
```

## **Creating Navigation.kt in nav package**

```
package com.project.pradyotprakash.flashchat.nav

import androidx.navigation.NavHostController

import com.project.pradyotprakash.flashchat.nav.Destination.Home

import com.project.pradyotprakash.flashchat.nav.Destination.Login

import com.project.pradyotprakash.flashchat.nav.Destination.Register

object Destination {

    const val AuthenticationOption = "authenticationOption"

    const val Register = "register"

    const val Login = "login"

    const val Home = "home"

}

class Action(navController: NavHostController) {
```

```

val home: () -> Unit = {

    navController.navigate(Home) {

        popUpTo(Login) {

            inclusive = true

        }

        popUpTo(Register) {

            inclusive = true

        }

    }

}

val login: () -> Unit = { navController.navigate(Login) }

val register: () -> Unit = { navController.navigate(Register) }

val navigateBack: () -> Unit = { navController.popBackStack() }

}

```

## Creating view package

```

package com.project.pradyotprakash.flashchat.view

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.fillMaxHeight

import androidx.compose.foundation.layout.fillMaxWidth

```

```

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.*

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import com.project.pradyotprakash.flashchat.ui.theme.FlashChatTheme

@Composable

fun AuthenticationView(register: () -> Unit, login: () -> Unit) {

    FlashChatTheme {

        // A surface container using the 'background' color from the theme

        Surface(color = MaterialTheme.colors.background) {

            Column(

                modifier = Modifier

                    .fillMaxWidth()

                    .fillMaxHeight(),

                horizontalAlignment = Alignment.CenterHorizontally,

                verticalArrangement = Arrangement.Bottom

            ) {

                Title(title = "⚡ Chat Connect")

                Buttons(title = "Register", onClick = register, backgroundColor = Color.Blue)

```

```
        Buttons(title = "Login", onClick = login, backgroundColor = Color.Magenta)

    }

}

}
```

## **Widgets.kt**

```
package com.project.pradyotprakash.flashchat.view

import androidx.compose.foundation.layout.fillMaxHeight

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.foundation.text.KeyboardOptions

import androidx.compose.material.*

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.ArrowBack

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.KeyboardType
```



```
import androidx.compose.ui.text.input.VisualTransformation
```

```
import androidx.compose.ui.text.style.TextAlign
```

```
import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp
```

```
import com.project.pradyotprakash.flashchat.Constants
```

```
@Composable
```

```
fun Title(title: String) {
```

```
    Text(
```

```
        text = title,
```

```
        fontSize = 30.sp,
```

```
        fontWeight = FontWeight.Bold,
```

```
        modifier = Modifier.fillMaxHeight(0.5f)
```

```
    )
```

```
}
```

```
// Different set of buttons in this page
```

```
@Composable
```

```
fun Buttons(title: String, onClick: () -> Unit, backgroundColor: Color) {
```

```
    Button(
```

```
        onClick = onClick,
```

```
        colors = ButtonDefaults.buttonColors(
```

```
        backgroundColor = backgroundColor,

        contentColor = Color.White

    ),

    modifier = Modifier.fillMaxWidth(),

    shape = RoundedCornerShape(0),

) {

    Text(

        text = title

    )

}

}

@Composable

fun AppBar(title: String, action: () -> Unit) {

    TopAppBar(

        title = {

            Text(text = title)

        },

        navigationIcon = {

            IconButton(

                onClick = action

            ) {
```

```

        Icon(

            imageVector = Icons.Filled.ArrowBack,

            contentDescription = "Back button"

        )

    }

}

)

}

@Composable

fun TextFormField(value: String, onValueChange: (String) -> Unit, label: String, keyboardType:

KeyboardType, visualTransformation: VisualTransformation) {

    OutlinedTextField(

        value = value,

        onValueChange = onValueChange,

        label = {

            Text(

                label

            )

        },

        maxLines = 1,

        modifier = Modifier

```

```
.padding(horizontal = 20.dp, vertical = 5.dp)
```

```
.fillMaxWidth(),
```

```
keyboardOptions = KeyboardOptions(
```

keyboardType = keyboardType

$$),$$

```
singleLine = true,
```

```
visualTransformation = visualTransformation
```

)

$$\}$$

@Composable

```
fun SingleMessage(message: String, isCurrentUser: Boolean) {
```

Card(

```
shape = RoundedCornerShape(16.dp),
```

```
backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else Color.White
```

 $) \{$ 

Text(

```
text = message,
```

textAlign =

```
if (isCurrentUser)
```

TextAlign.End

else

```

        TextAlign.Start,

        modifier = Modifier.fillMaxWidth().padding(16.dp),

        color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White

    )

}

}

```

## Creating home package in view package

```

package com.project.pradyotprakash.flashchat.view.home

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.*

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.text.KeyboardOptions

import androidx.compose.material.*

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.Send

import androidx.compose.runtime.Composable

import androidx.compose.runtime.getValue

import androidx.compose.runtime.livedata.observeAsState

import androidx.compose.ui.Alignment

```

```
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.text.input.KeyboardType
```

```
import androidx.compose.ui.unit.dp
```

```
import androidx.lifecycle.viewmodel.compose.viewModel
```

```
import com.project.pradyotprakash.flashchat.Constants
```

```
import com.project.pradyotprakash.flashchat.view.SingleMessage
```

```
@Composable
```

```
fun HomeView(
```

```
    homeViewModel: HomeViewModel = viewModel()
```

```
) {
```

```
    val message: String by homeViewModel.message.observeAsState(initial = "")
```

```
    val messages: List<Map<String, Any>> by homeViewModel.messages.observeAsState(
```

```
        initial = emptyList<Map<String, Any>>().toMutableList()
```

```
)
```

```
Column(
```

```
    modifier = Modifier.fillMaxSize(),
```

```
    horizontalAlignment = Alignment.CenterHorizontally,
```

```
    verticalArrangement = Arrangement.Bottom
```

```
) {
```

```

LazyColumn(

    modifier = Modifier

        .fillMaxWidth()

        .weight(weight = 0.85f, fill = true),

    contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),

    verticalArrangement = Arrangement.spacedBy(4.dp),

    reverseLayout = true

) {

    items(messages) { message ->

        val isCurrentUser = message[Constants.IS_CURRENT_USER] as Boolean

        SingleMessage(

            message = message[Constants.MESSAGE].toString(),

            isCurrentUser = isCurrentUser

        )

    }

}

OutlinedTextField(

    value = message,

    onChange = {

        homeViewModel.updateMessage(it)
    }
)

```

```
    },  
  
    label = {  
  
        Text(  
  
            "Type Your Message"  
  
        )  
  
    },  
  
    maxLines = 1,  
  
    modifier = Modifier  
  
        .padding(horizontal = 15.dp, vertical = 1.dp)  
  
        .fillMaxWidth()  
  
        .weight(weight = 0.09f, fill = true),  
  
    keyboardOptions = KeyboardOptions(  
  
        keyboardType = KeyboardType.Text  
  
    ),  
  
    singleLine = true,  
  
    trailingIcon = {  
  
        IconButton(  
  
            onClick = {  
  
                homeViewModel.addMessage()  
  
            }  
  
        ) {
```



```

        Icon(

            imageVector = Icons.Default.Send,

            contentDescription = "Send Button"

        )

    }

}

)

}

}

```

## Creating HomeViewModel class

```

package com.project.pradyotprakash.flashchat.view.home

import android.util.Log

import androidx.lifecycle.LiveData

import androidx.lifecycle.MutableLiveData

import androidx.lifecycle.ViewModel

import com.google.firebase.auth.ktx.auth

import com.google.firebase.firestore.ktx.firestore

import com.google.firebase.ktx.Firebase

import com.project.pradyotprakash.flashchat.Constants

import java.lang.IllegalArgumentException

class HomeViewModel : ViewModel() {

    init {

        getMessages()

    }

}

```

```
private val _message = MutableLiveData("")
```

```
val message: LiveData<String> = _message
```

```
private var _messages = MutableLiveData(emptyList<Map<String, Any>>().toMutableList())
```

```
val messages: LiveData<MutableList<Map<String, Any>>> = _messages
```

```
/**
```

```
 * Update the message value as user types
```

```
 */
```

```
fun updateMessage(message: String) {
```

```
    _message.value = message
```

```
}
```

```
/**
```

```
 * Send message
```

```
 */
```

```
fun addMessage() {
```

```
    val message: String = _message.value ?: throw IllegalArgumentException("message empty")
```

```
    if (message.isNotEmpty()) {
```

```
        Firebase.firestore.collection(Constants.MESSAGES).document().set(
```

```
            hashMapOf(
```

```
                Constants.MESSAGE to message,
```

```
                Constants.SENT_BY to Firebase.auth.currentUser?.uid,
```

```
                Constants.SENT_ON to System.currentTimeMillis()
```

```
            )
```

```

        ).addOnSuccessListener {
            _message.value = ""
        }
    }
}

```

```

/**

```

```

 * Get the messages

```

```

 */

```

```

private fun getMessages() {

```

```

    Firebase.firestore.collection(Constants.MESSAGES)

```

```

        .orderBy(Constants.SENT_ON)

```

```

        .addSnapshotListener { value, e ->

```

```

            if (e != null) {

```

```

                Log.w(Constants.TAG, "Listen failed.", e)

```

```

                return@addSnapshotListener

```

```

            }

```

```

        val list = emptyList<Map<String, Any>>().toMutableList()

```

```

        if (value != null) {

```

```

            for (doc in value) {

```

```

                val data = doc.data

```

```

                data[Constants.IS_CURRENT_USER] =

```

```

                    Firebase.auth.currentUser?.uid.toString() == data[Constants.SENT_BY].toString()

```

```

        list.add(data)
    }
}

updateMessages(list)
}
}

/**
 * Update the list after getting the details from firestore
 */
private fun updateMessages(list: MutableList<Map<String, Any>>) {
    _messages.value = list.asReversed()
}
}

```

## Creating login package in view package

```

package com.project.pradyotprakash.flashchat.view.login

import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

```

```

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.text.input.KeyboardType

import androidx.compose.ui.text.input.PasswordVisualTransformation

import androidx.compose.ui.text.input.VisualTransformation

import androidx.compose.ui.unit.dp

import androidx.lifecycle.viewmodel.compose.viewModel

import com.project.pradyotprakash.flashchat.view.Appbar

import com.project.pradyotprakash.flashchat.view.Buttons

import com.project.pradyotprakash.flashchat.view.TextFormField

@Composable

fun LoginView(

    home: () -> Unit,

    back: () -> Unit,

    loginViewModel: LoginViewModel = viewModel()

) {

    val email: String by loginViewModel.email.observeAsState("")

    val password: String by loginViewModel.password.observeAsState("")

    val loading: Boolean by loginViewModel.loading.observeAsState(initial = false)

    Box(

        contentAlignment = Alignment.Center,

        modifier = Modifier.fillMaxSize()

    ) {

        if (loading) {

            CircularProgressIndicator()

        }

    }

```

```
Column(  
    modifier = Modifier.fillMaxSize(),  
    horizontalAlignment = Alignment.CenterHorizontally,  
    verticalArrangement = Arrangement.Top  
) {  
    AppBar(  
        title = "Login",  
        action = back  
    )  
    TextFormField(  
        value = email,  
        onValueChange = { loginViewModel.updateEmail(it) },  
        label = "Email",  
        keyboardType = TextInputType.Email,  
        visualTransformation = VisualTransformation.None  
    )  
    TextFormField(  
        value = password,  
        onValueChange = { loginViewModel.updatePassword(it) },  
        label = "Password",  
        keyboardType = TextInputType.Password,  
        visualTransformation = PasswordVisualTransformation()  
    )  
    Spacer(modifier = Modifier.height(20.dp))  
    Buttons(  
        title = "Login",
```

```

        onClick = { loginViewModel.loginUser(home = home) },

        backgroundColor = Color.Magenta
    )
}
}
}

```

## **package com.project.pradyotprakash.flashchat.view.login**

```

import androidx.lifecycle.LiveData

import androidx.lifecycle.MutableLiveData

import androidx.lifecycle.ViewModel

import com.google.firebase.auth.FirebaseAuth

import com.google.firebase.auth.ktx.auth

import com.google.firebase.ktx.Firebase

import java.lang.IllegalArgumentException

class LoginViewModel : ViewModel() {

    private val auth: FirebaseAuth = Firebase.auth

    private val _email = MutableLiveData("")

    val email: LiveData<String> = _email

    private val _password = MutableLiveData("")

    val password: LiveData<String> = _password

    private val _loading = MutableLiveData(false)

```

```
val loading: LiveData<Boolean> = _loading
```

```
// Update email
```

```
fun updateEmail(newEmail: String) {  
    _email.value = newEmail  
}
```

```
// Update password
```

```
fun updatePassword(newPassword: String) {  
    _password.value = newPassword  
}
```

```
// Register user
```

```
fun loginUser(home: () -> Unit) {  
    if (_loading.value == false) {  
        val email: String = _email.value ?: throw IllegalArgumentException("email expected")  
        val password: String =  
            _password.value ?: throw IllegalArgumentException("password expected")  
  
        _loading.value = true  
  
        auth.signInWithEmailAndPassword(email, password)  
            .addOnCompleteListener {  
                if (it.isSuccessful) {  
                    home()  
                }  
            }  
    }  
}
```



```
        _loading.value = false
    }
}
}
```

## Creating register package in view package

```
package com.project.pradyotprakash.flashchat.view.register

import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.project.pradyotprakash.flashchat.view.Appbar
import com.project.pradyotprakash.flashchat.view.Buttons
import com.project.pradyotprakash.flashchat.view.TextFormField

@Composable
```

```

fun RegisterView(

    home: () -> Unit,

    back: () -> Unit,

    registerViewModel: RegisterViewModel = viewModel()

) {

    val email: String by registerViewModel.email.observeAsState("")

    val password: String by registerViewModel.password.observeAsState("")

    val loading: Boolean by registerViewModel.loading.observeAsState(initial = false)

    Box(

        contentAlignment = Alignment.Center,

        modifier = Modifier.fillMaxSize()

    ) {

        if (loading) {

            CircularProgressIndicator()

        }

        Column(

            modifier = Modifier.fillMaxSize(),

            horizontalAlignment = Alignment.CenterHorizontally,

            verticalArrangement = Arrangement.Top

        ) {

            AppBar(

                title = "Register",

                action = back

            )

            TextFormField(

```

```

        value = email,

        onValueChange = { registerViewModel.updateEmail(it) },

        label = "Email",

        keyboardType = KeyboardType.Email,

        visualTransformation = VisualTransformation.None
    )

    TextFormField(

        value = password,

        onValueChange = { registerViewModel.updatePassword(it) },

        label = "Password",

        keyboardType = KeyboardType.Password,

        visualTransformation = PasswordVisualTransformation()
    )

    Spacer(modifier = Modifier.height(20.dp))

    Buttons(

        title = "Register",

        onClick = { registerViewModel.registerUser(home = home) },

        backgroundColor = Color.Blue
    )
}

}

}

```

## Creating RegisterViewModel class

```

package com.project.pradyotprakash.flashchat.view.register

import androidx.lifecycle.LiveData

```

```
import androidx.lifecycle.MutableLiveData

import androidx.lifecycle.ViewModel

import com.google.firebase.auth.FirebaseAuth

import com.google.firebase.auth.ktx.auth

import com.google.firebase.ktx.Firebase

import java.lang.IllegalArgumentException

class RegisterViewModel : ViewModel() {

    private val auth: FirebaseAuth = Firebase.auth

    private val _email = MutableLiveData("")

    val email: LiveData<String> = _email

    private val _password = MutableLiveData("")

    val password: LiveData<String> = _password

    private val _loading = MutableLiveData(false)

    val loading: LiveData<Boolean> = _loading

    // Update email

    fun updateEmail(newEmail: String) {

        _email.value = newEmail

    }

    // Update password

    fun updatePassword(newPassword: String) {

        _password.value = newPassword

    }

}
```

```
}
```

```
// Register user
```

```
fun registerUser(home: () -> Unit) {
```

```
    if (_loading.value == false) {
```

```
        val email: String = _email.value ?: throw IllegalArgumentException("email expected")
```

```
        val password: String =
```

```
            _password.value ?: throw IllegalArgumentException("password expected")
```

```
        _loading.value = true
```

```
        auth.createUserWithEmailAndPassword(email, password)
```

```
        .addOnCompleteListener {
```

```
            if (it.isSuccessful) {
```

```
                home()
```

```
            }
```

```
            _loading.value = false
```

```
        }
```

```
    }
```

```
}
```

```
}
```