

# **A REAL TIME CHAT AND COMMUNICATON APP**

**By:**

**Team Leader: E.Shunmuga priya**

**Team Member's: S. Mutheeswari**

**A.V. Sobhana**

**R.N. Thangam**

# INTRODUCTION

## Overview:

- Chatting apps are software tools that allow users to send and receive messages online.
- This kind of application is built into every smartphone, it is available for everyone and it is very popular.
- A chat app refers to a text or video messaging application.
- Chat connect is a real-time chat messaging tool that enables users to chat with individuals and groups.
- It is very convenient.
- Communication apps are more effective way to reach everyone in an organization.
- Encouraging members to make decision as a group.
- Real-Time messaging features enables user to have authentic and interactive experience.

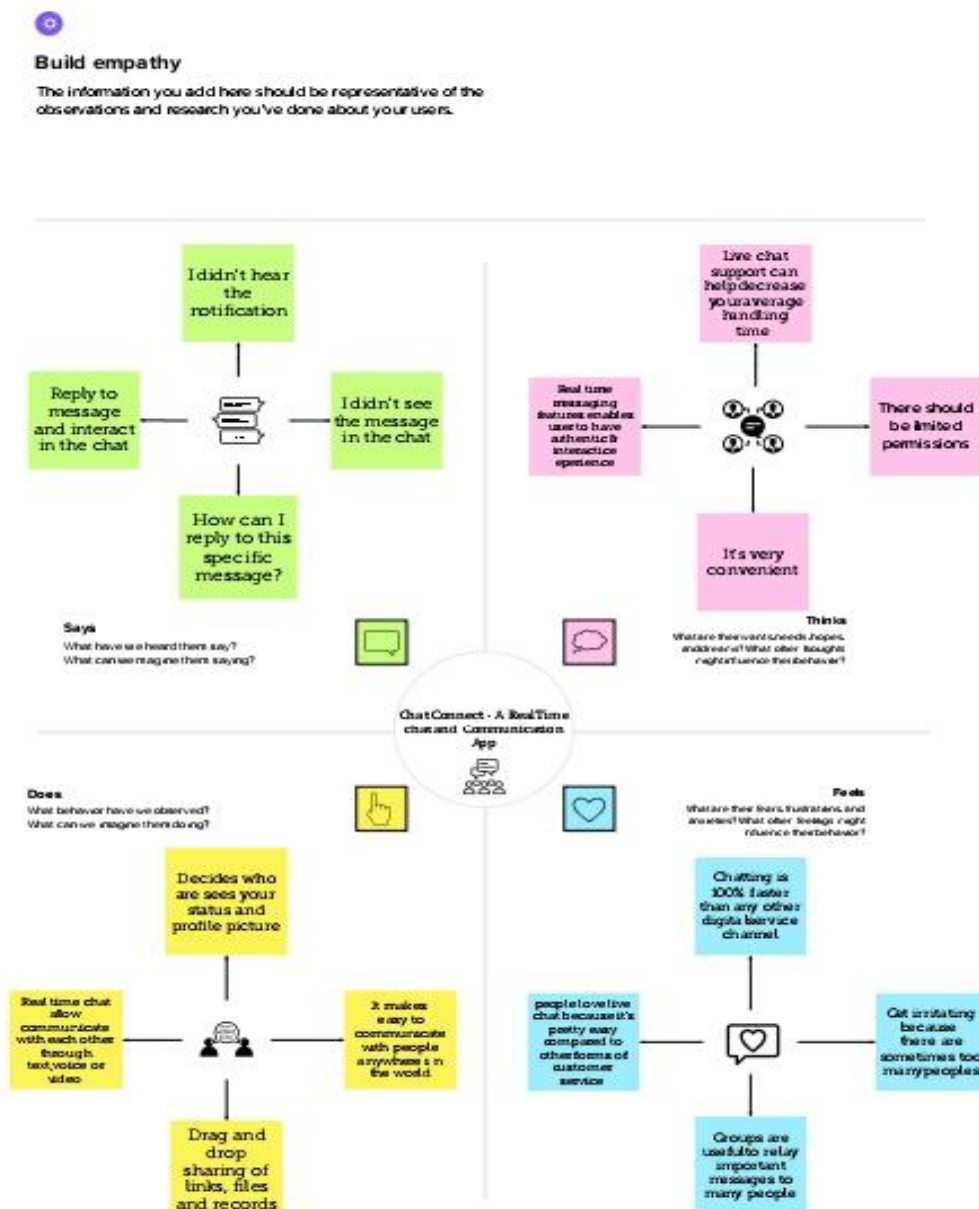
## **Purpose:**

- A chat application makes it easy to communicate with people anywhere in the world by sending and receiving messages in real time.
- With a web or mobile chat app, users are able to receive the same engaging and lively interactions through custom messaging features, just as they would in person.
- Online chats have become an eminent part of both our daily life and business environment.
- Instant messaging is faster and more convenient than email and less stressful than a phone call, so there is no wonder why we use chat apps so often.
- People around the world are used to chatting online about everything, including discussing the latest episode with a friend, organizing a team-building meeting.

# Problem Definition and Design Thinking

=

## Empathy Map:



## Ideation and Brainstorming Map:

E.Shunmuga Priya	AV.Sobhana	R.NThangam	S.Mutheeswari
Its allow user to communicate with each other in real-time through text, voice or videos.	Creating a way for the team to build rapport with one another.	Direct communication between two peoples.	It is useful for students to talk to each other without meeting each other.
Establishing mutual respect and understanding among group members.	A Real-time messaging tool that enables user to chat with individuals & groups quickly share files.	Encouraging member to make decisions as a group	Datas are leaks in the real-time chat
It provides drag-and-drop sharing files, records and links	If you don't understand any lesson conducted in school, you can share and ask through this.	Communication apps are the most effective way to reach everyone in an organization.	Providing a way for group members to generate ideas and solutions to achieve their common goal.
You can't be sure other people are being honest or that they are who they say they are.	A Digital chat app that's composed of realtime messaging futures enables users to have authentic and interactive experience.	Live chat reduces repetition for your customer.	Quickly share files and collaborate on any record by collecting with the right people instantly.

## **Important Ideas**

*It allows user to communicate with each other in real-time through text, video or voice.*

*It is useful for students to talk to each other without meeting each other.*

*Direct communication between two users.*

*Communication app are the most effective way to reach everyone in an organization.*

*It provides drag and drop sharing files, records and links*


*A Real-Time messaging tool that enables user to chat with individuals and groups quickly share files.*

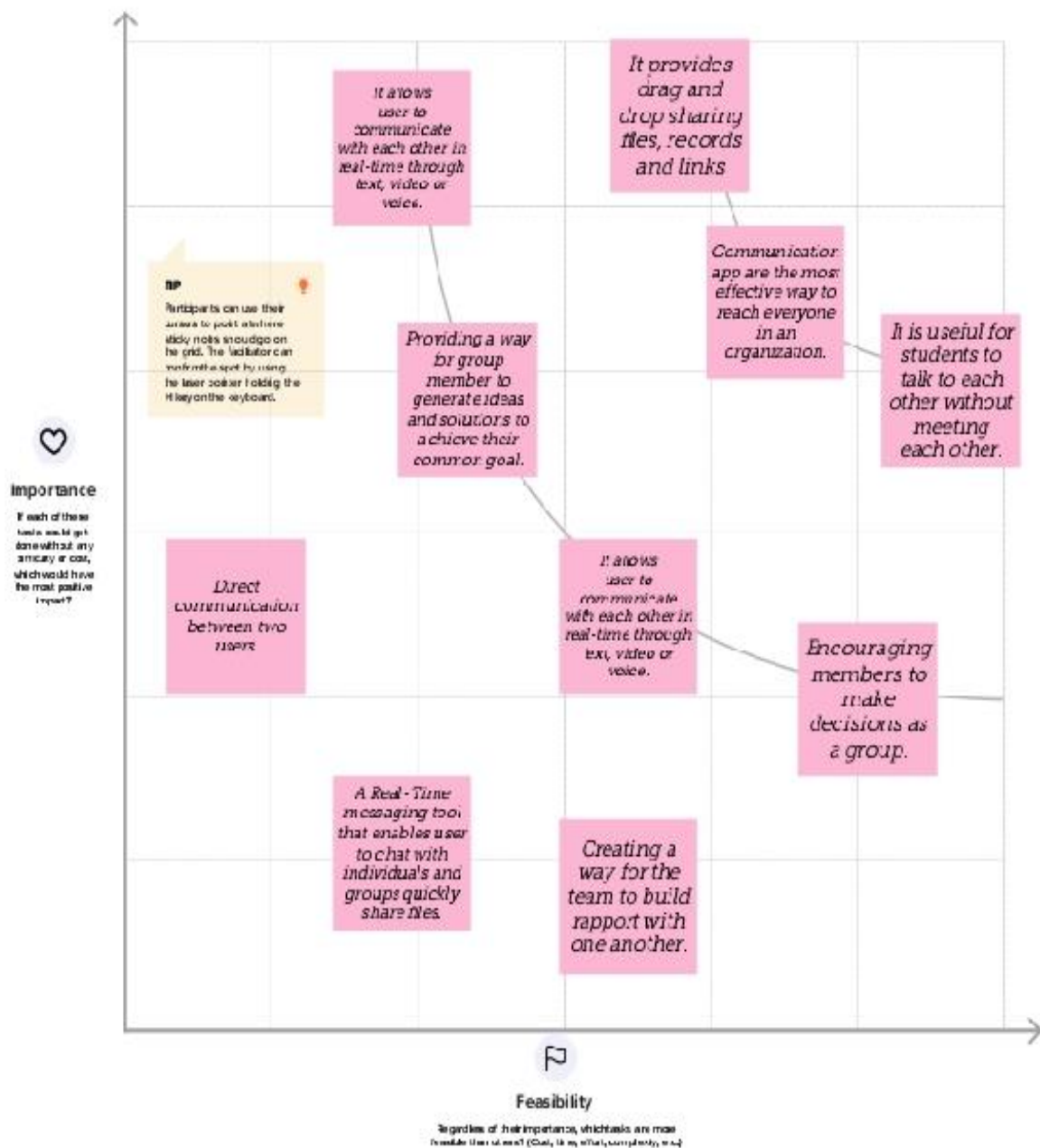
*Encouraging members to make decisions as a group.*

*Creating a way for the team to build rapport with one another.*

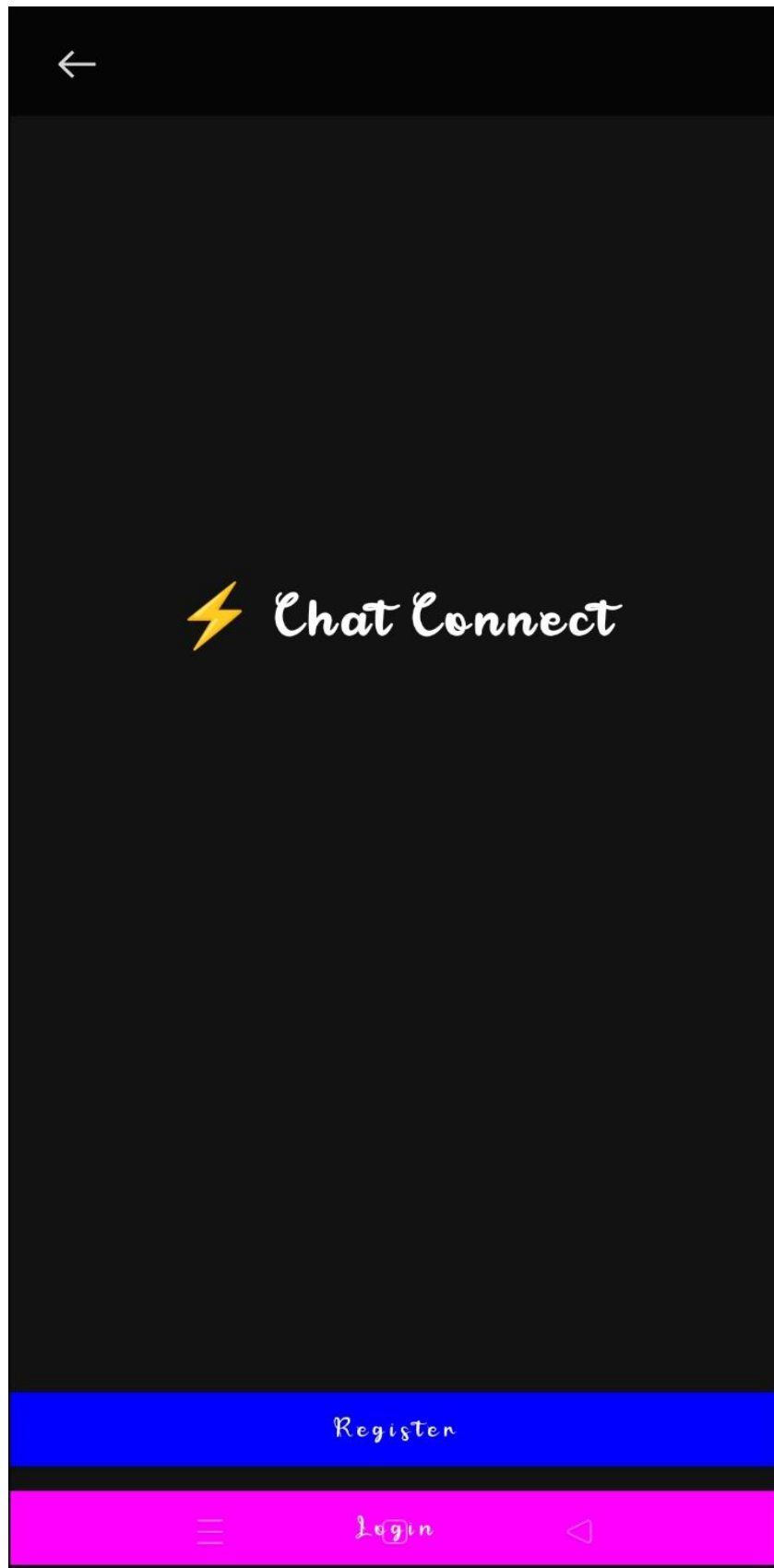
*Providing a way for group member to generate ideas and solutions to achieve their common goal.*

### **TIP**

 Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.



# RESULT







Register

Email

Password

Register



Login

Email

Password

Login

hi

hlo

hi

hi

hi

hlo

Hf

hii



hlo

how are you

nice to meet you

Type Your Message

|



# ADVANTAGES AND DISADVANTAGES

## ADVANTAGES:

- It involves the transmission of live text messages from the sender to the recipient.
- Drag-and-Drop sharing of links, files and records.
- Online chatting can boost the confidence and self-esteem of individuals who have heartaches or feel lonely.
- Online dating is becoming increasingly common, as some people are too busy to go out and meet someone.
- If someone feels uncomfortable chatting with a stranger, it is easy to leave.
- Online chatting can create, re-create or maintain relationships, despite being oceans apart.
- The chance to meet new people with different views.

## DISADVANTAGE:

- You can't be sure other people are being honest or that they are who they say they are.
- If you are feeling vulnerable. People online might try to take advantage of you.
- Building relationships online can result in your spending less time with friends and family.
- Dates are links in the real-time chat.
- Depending on the service, you could receive unwanted calls or calls from people you do not know.
- Hacking and Cheating.

# APPLICATIONS

- It is useful for students to talk to each other without meeting each other.
- It is used for online classes.
- It allows you to communicate with your customers in web chat rooms.
- Chat connect app is used for business, if there is an urgent meeting it is easy way to call the team members immediately.
- Chatting app is used in online shopping, because a customer want to know the quality of product. So he ask with seller to know the detailed information about the product.
- The Family members are in long distance, but this app connect everyone in a particular place.

# **CONCLUSION**

- Chatting app are software tools that allow users to send and receive messages online.
- In this application easy to communicate with each other.
- It contains several advantages and disadvantages of chat connect.
- Online chats have become an eminent part of both our daily life and business environment.
- Drag-and-Drop sharing of links, files and records.
- You can't be sure other people are being honest or that they are who they say they are.
- We don't know who is on the other side to communicate with each other, this is a main disadvantage of chat connect.
- 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, group chat, etc.
- The main objective of the project is to develop a Secure Chat Application.
- I had taken a wide range of literature review in order to achieve all the tasks, where I came to know about some of the products that are existing in the market.
- I made a detailed research in that path to cover the loop holes that existing systems are facing and to eradicate them in our application.
- In the process of research I came to know about the latest technologies and different algorithms.
- I analysed various encryption algorithms (DES, AES, IDEA...), Integrity algorithms (MD5, SHA), key-exchange algorithms, authentication and I had implemented those functionalities in my application.



## 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 Voice Chat.
- Extending it to Web Support.

## **1. ENCRYPTION & ONLINE PRIVACY**

Starting from terabytes of data sent via your messenger, and ending with confidential data of the large corporations which are loaded, transferred and stored every single day thus, respect for privacy and adequate data protection are critical.

- New advanced features for messenger guarantee all your messages and their receivers have the keys necessary to decode their content and make it very difficult for any third party to read your sms.

## **2. CLOUD & DATA SYNCHRONIZATION**

Cloud Services are hardly new features on messenger enabled storing your files in different places up-to-date.

- All other devices are updated automatically when you make a modification to a document on one device.

### **DATA PERSISTENCE FOR OFFLINE USAGE**

- These new features in the messenger platform save your battery.
- WiFi and mobile Internet eat your battery.
- The mobile application with full support of offline mode is escalating at a higher rate.

**Your app's offline mode offers several benefits:**

- No roaming cost when you are traveling as all the files can be cache.
- No monthly data usage on maps
- Quick loading time

## **4. BOTS**

These are small special features of instant messaging software that are embedded in chats or public channels to perform a specific function.

What is its purpose?

- Self-destructing statuses may be used to manage communities and blogs.
- Automated assistance with registration and other concerns.
- Custom created chat bots using the open-source frameworks, etc.

## **5. LIGHT MODE AND DARK MODE**

- These are not only desirable features of a good message passing system.
- Additionally, real improvements in battery life may be gained.
- Google claims that accessing YouTube in Dark Mode saves batteries by 14 percent when the screen brightness is set to 50 percent.

# APPENDIX

Source code:

MainActivity.kt

```
package com.project.pradyotprakash.flashchat

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.FirebaseApp

/**
 * The initial point of the application from where it gets
 * started.
 *
 * Here we do all the initialization and other things which will
 * be required
 * thought out the application.
 */
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        FirebaseApp.initializeApp(this)
        setContent {
            NavComposeApp()
        }
    }
}
```

## **NavComposeApp.kt**

```
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.Regist
er

import
com.project.pradyotprakash.flashchat.ui.theme.FlashChatTh
eme

import
com.project.pradyotprakash.flashchat.view.AuthenticationVi
ew

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

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

import
com.project.pradyotprakash.flashchat.view.register.Register
View
```

```
/**
```

```
 * The main Navigation composable which will handle all the
navigation stack.
```

```
 */
```

```
@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()
}
}
}
}
```

## **Constants.kt**

```
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"
}
```

## **Navigation.kt**

```
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.Regist
er

/**
```

\* A set of destination used in the whole application

\*/

object Destination {

    const val AuthenticationOption = "authenticationOption"

    const val Register = "register"

    const val Login = "login"

    const val Home = "home"

}

/\*\*

\* Set of routes which will be passed to different composable so that

\* the routes which are required can be taken.

\*/

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() }  
}
```

## **AuthenticationOption.kt**

```
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.FlashChatTh
eme
```

```
/**
```

```
 * The authentication view which will give the user an option
to choose between
```

```
 * login and register.
```

```
*/
```

```
@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
```

```
/**
```

```
 * Set of widgets/views which will be used throughout the
application.
```

```
 * This is used to increase the code usability.
```

```
*/
```

```
@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  
        )  
    }  
}
```

```
)  
}  
}
```

## Home.kt

```
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
```

```
/**
```

```
 * The home view which will contain all the code related to
the view for HOME.
```

```
 *
```

```
 * Here we will show the list of chat messages sent by user.
```

```
 * And also give an option to send a message and logout.
```

```
 */
```

```
@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,  
    onValueChange = {  
        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"
        )
    }
}
)
}
```

# Home.kt

```
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
```

```
/**
```

```
 * The home view which will contain all the code related to
the view for HOME.
```

```
 *
```

```
 * Here we will show the list of chat messages sent by user.
```

```
 * And also give an option to send a message and logout.
```

```
 */
```

```
@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"
        )
    }
}
)
}
```

## **HomeViewModel.kt**

```
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

/**
 * Home view model which will handle all the logic related to
 * HomeView
 */
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()  
        t().set(  
            hashMapOf(  
                Constants.MESSAGE to message,  
            )  
        )  
    }
```

```

        Constants.SENT_BY to
        FirebaseAuth.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()
    }
}
```

## Login.kt

```
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.PasswordVisualTransformati
on
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
```

```
/**
```

```
 * The login view which will help the user to authenticate
themselves and go to the
```

```
 * home screen to show and send messages to others.
```

```
*/
```

```
@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 = KeyboardType.Email,
    visualTransformation = VisualTransformation.None
)
TextFormField(
    value = password,
    onValueChange = {
loginViewModel.updatePassword(it) },
    label = "Password",
    keyboardType = KeyboardType.Password,
    visualTransformation =
PasswordVisualTransformation()
)
Spacer(modifier = Modifier.height(20.dp))
Buttons(
    title = "Login",
    onClick = { loginViewModel.loginUser(home = home)
},
    backgroundColor = Color.Magenta
)
}
```

```
}  
}
```

## **LoginViewModel.kt**

```
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
```

```
/**
```

```
 * View model for the login view.
```

```
*/
```

```
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
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

## Register.kt

```
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
```

```
/**
```

```
 * The Register view which will be helpful for the user to  
register themselves into
```

```
 * our database and go to the home screen to see and send  
messages.
```

```
*/
```

```
@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 = TextInputType.Email,
```

```
        visualTransformation = VisualTransformation.None
    )
    TextFormField(
        value = password,
        onValueChange = {
registerViewModel.updatePassword(it) },
        label = "Password",
        keyboardType = TextInputType.Password,
        visualTransformation =
PasswordVisualTransformation()
    )
    Spacer(modifier = Modifier.height(20.dp))
    Buttons(
        title = "Register",
        onClick = { registerViewModel.registerUser(home =
home) },
        backgroundColor = Color.Blue
    )
    }
}
}
```

## **RegisterViewModel.kt**

```
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
```

```
/**
```

```
 * View model for the login view.
```

```
*/
```

```
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
            }
        }
    }
}
```