**CHAT CONNECT-A REAL TIME CHAT AND COMMUNICATION APP**

**PROJECT REPORT**

***Submitted by***

**REJIN R J (20203111506240)**

**RAHUL C K(20203111506239)**

**RISBIN R H(20203111506241)**

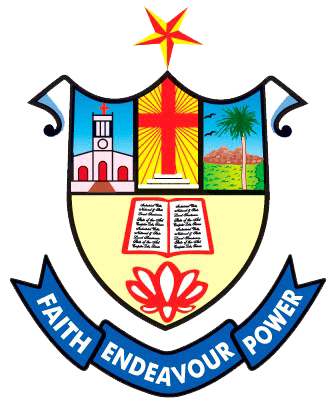
**PRITHVI M R (20203111506237)**

***Submitted to Manonmaniam Sundaranar University. Tirunelveli  
 In partial fulfilment for the award of***

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

***Under the Guidance of***

**Prof. D. H. KITTY SMAILIN M.Sc., M. Phill**



**DEPARTMENT OF PG COMPUTER SCIENCE**

**NESAMONY MEMORIAL CHRISTIAN COLLEGE,**

**MARTHANDAM**

**KANYAKUMARI DISTRICT -629165**

***Re-accredited with ‘A’ grade by NAAC***

**MARCH 2023**

**DECLARATION**

I hereby declare that the project work entitled “**CHAT CONNECT- A REAL TIME CHAT AND COMMUNICATION APP”** is an original work done by me in partial fulfilment of the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE.** The study has been carried under the guidance of  **Prof.D.H.KITTY SMAILIN M.Sc.,M.Phil.** **Department of PG COMPUTER SCIENCE, Nesamony Memorial Christian College, Marthandam.** I declare that this work has not been submitted elsewhere for the award of any degree.

Place: Marthandam. Rejin R J (20203111506240)

Rahul C K (20203111506239 )

Risbin R H (20203111506241)

Prithivi M R(20203111506237)

**ACKNOWLEDGEMENT**

First of all I offer my prayers to god almighty for blessing me to complete this internship programme successfully

I express my sincere thanks to **, Dr.K.Paul Raj M.Sc., M.Phil., M.Ed., M.Phil(Edu).,Ph.D.,** Principal Nesamony Memorial Christian college, for his official support to do my work.

I express my profound thanks to **Dr. Dr.D.Latha M.Sc., M.Phil., Ph.D.,** HOD, Department of PG Computer Science for his encouragement given to undertake this project.

I extend my deep sense of gratitude to **, Prof**. **D.H.KITTY SMAILIN M.Sc.,M.Phil** Department of of PG Computer Science for guiding me in completing this project work successfully.

My special thanks to other Faculty members of Department of

PG Computer Science for Guiding me in Completing this project.

I also wish to extend a special word of thanks to my parents, friends and all unseen hands that helped me for completing this project work successfully.

**1.INTRODUCTION**

**1.1 OVERVIEW**

A chatting app is a software application that allows users to communicate with each other in real-time through text messages, voice or video calls, and multimedia content such as photos and videos. Chatting apps typically require an internet connection, and users can access them on a variety of devices, including smartphones, tablets, and computers. Some popular examples of chatting apps include WhatsApp, Facebook Messenger, iMessage, and Skype. These apps offer features such as group chatting, end-to-end encryption, and the ability to share media files, making them a convenient and popular way for people to stay connected with friends, family, and colleagues.

**1.2 PURPOSE**

The use of this project. What can be achieved using this.

Stay connected with friends and family: Chatting apps allow people to stay in touch with their loved ones, regardless of where they are located. Users can exchange text messages, voice or video calls, and photos or videos, which helps them feel connected and updated with each other's lives.

Business communication: Chatting apps are also used for professional purposes, allowing co-workers and colleagues to communicate easily and efficiently, share files and documents, and coordinate on projects.

Social networking: Many chatting apps, such as Facebook Messenger, offer social networking features that allow users to connect with new people, join groups, and discover new content.

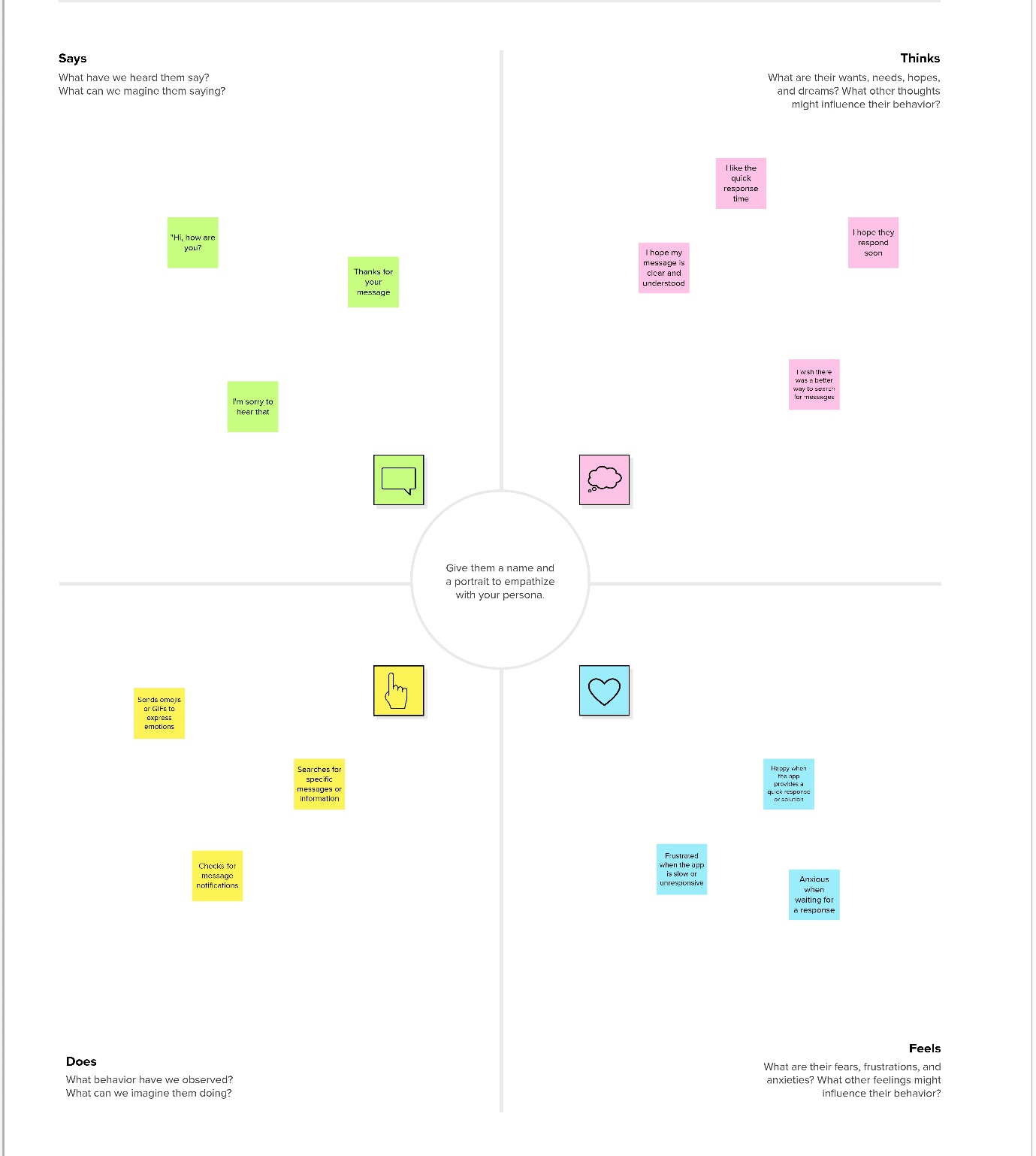
Entertainment: Chatting apps also provide a source of entertainment, with features such as emojis, GIFs, and stickers that add fun and personality to conversations.

Customer service: Many businesses use chatting apps as a customer service channel, providing a quick and convenient way for customers to ask questions, make requests, or file complaints.

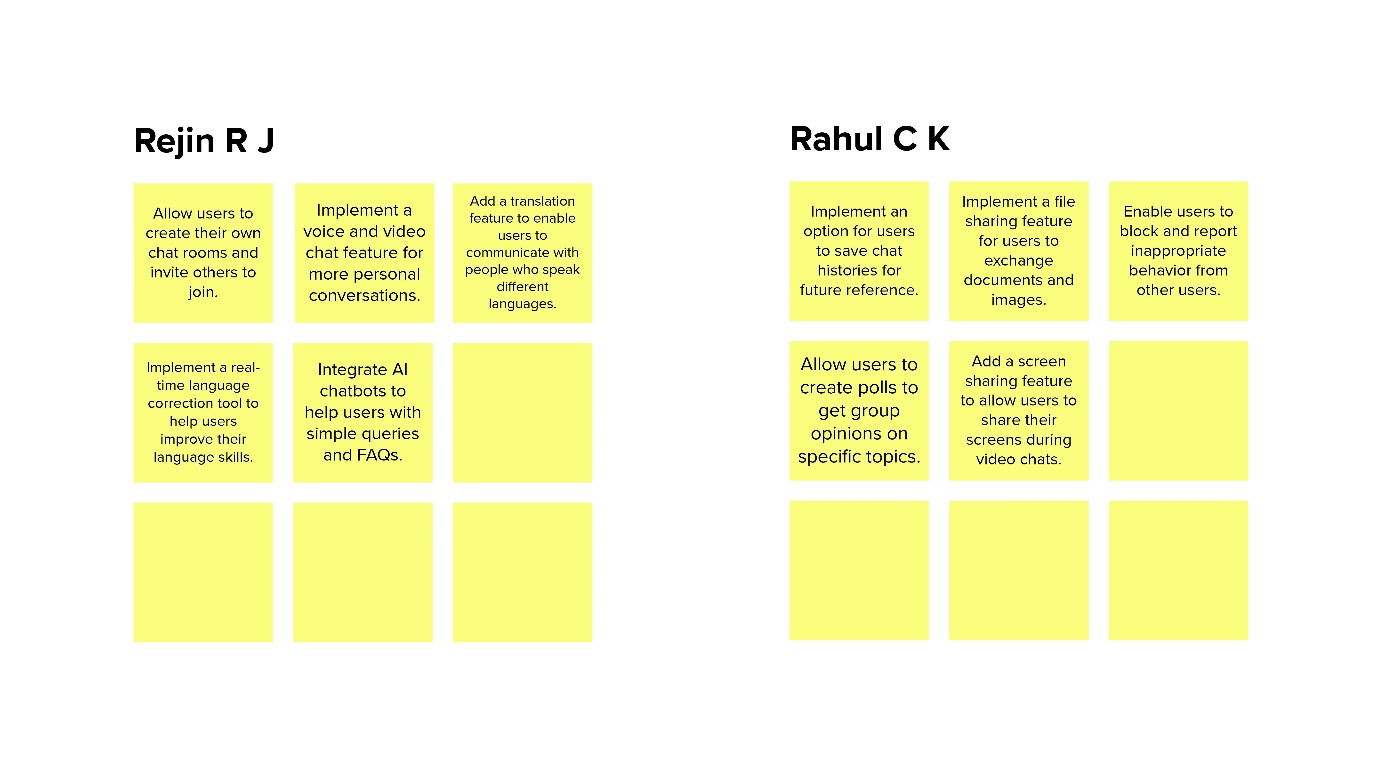
Overall, the use of chatting apps has revolutionized the way people communicate and connect with each other, making it easier, faster, and more convenient to stay in touch with others.

**Problem Definition & Design Thinking**

**2.1 Empathy Map**

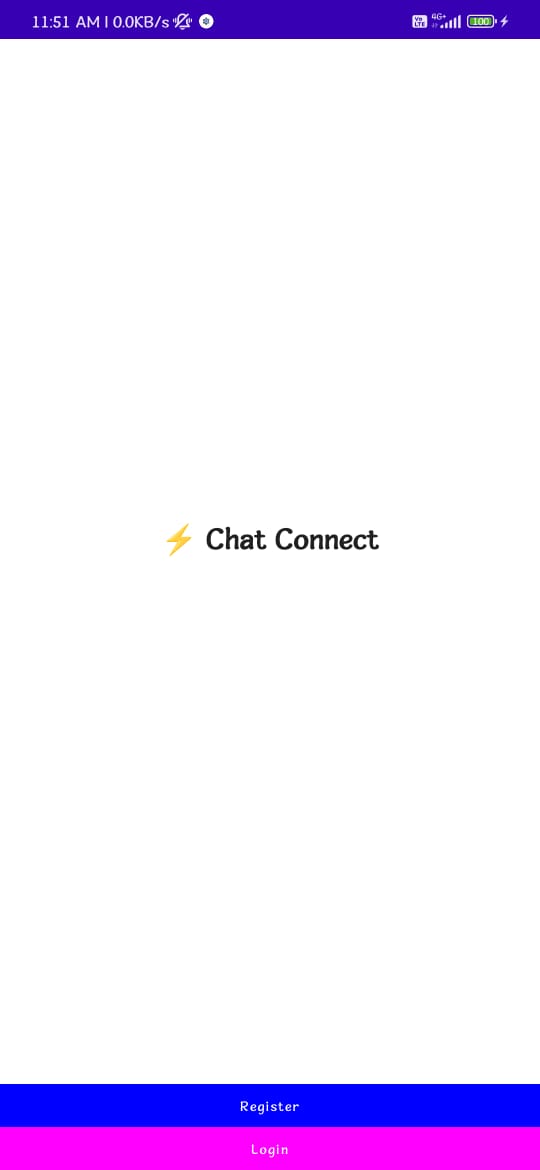


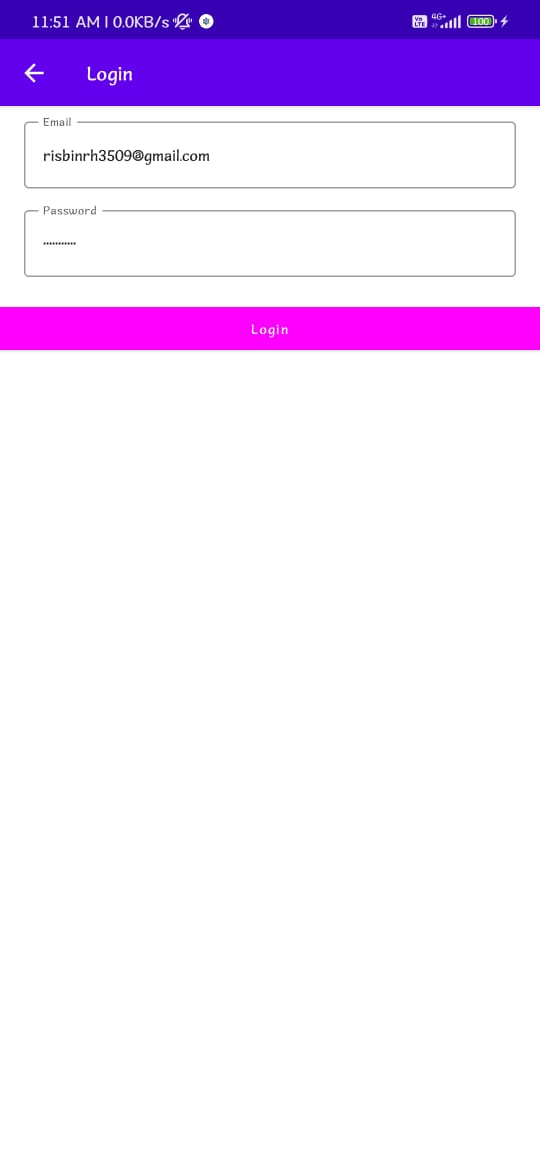
**2.2 Ideation & Brainstorming Map**

****

****

**3.RESULT**

****

****

****

**4.ADVANTAGES & DISADVANTAGES**

Advantages of chatting apps:

Instant communication: Chatting apps allow you to communicate with others instantly, regardless of the distance between you.

Cost-effective: Most chatting apps are free to use, which makes them a cost-effective way to communicate with others, especially for long-distance communication.

Convenience: Chatting apps are very convenient to use, as they can be accessed from anywhere, at any time, as long as you have an internet connection.

Group communication: Chatting apps allow you to communicate with multiple people at the same time, making them ideal for group communication.

Rich media sharing: Chatting apps often support sharing of various types of media, including photos, videos, voice notes, and documents, making it easy to share information and collaborate with others.

Disadvantages of chatting apps:

Security and privacy concerns: Chatting apps can be vulnerable to security breaches and privacy violations, especially if they are not properly secured and encrypted.

Addiction: Chatting apps can be addictive, leading to overuse and a loss of productivity.

Misinformation: Chatting apps can be used to spread misinformation and fake news, which can be harmful to individuals and society as a whole.

Decreased social interaction: Chatting apps can lead to decreased face-to-face social interaction, which can have negative effects on mental health and well-being.

Distractions: Chatting apps can be distracting, leading to a loss of focus and reduced productivity, especially when used during work or study time.

**5.APPLICATIONS**

The areas where this solution can be applied

The solution of using chatting apps responsibly and balancing their benefits and drawbacks can be applied in many areas where chatting apps are commonly used. For example:

Business: Many businesses use chatting apps for communication and collaboration among employees, teams, and clients. By using these apps responsibly, businesses can enhance their productivity and efficiency.

Education: Chatting apps are commonly used in education to facilitate communication between students and teachers, as well as for collaborative projects. By using these apps responsibly, educators can enhance student learning and engagement.

Healthcare: Chatting apps can be used in healthcare for communication between doctors, nurses, and patients, as well as for telemedicine consultations. By using these apps responsibly, healthcare providers can improve patient outcomes and access to care.

Social Media: Chatting apps are widely used on social media platforms for personal communication and social networking. By using these apps responsibly, individuals can enhance their social connections while avoiding the negative impacts of addiction, misinformation, and decreased social interaction.

In all of these areas, the responsible use of chatting apps can lead to significant benefits while minimizing the risks associated with their use.

**6.CONCLUSION**

In conclusion, this work has examined the advantages and disadvantages of chatting apps and the importance of using them responsibly. Chatting apps have revolutionized communication by offering instant messaging, cost-effectiveness, convenience, group communication, and media sharing. However, they also present some risks, such as security and privacy concerns, addiction, misinformation, decreased social interaction, and distractions. To maximize the benefits of chatting apps while minimizing their potential negative impacts, it is crucial to use them responsibly and balance their benefits and drawbacks.

This solution of using chatting apps responsibly can be applied in various areas such as business, education, healthcare, and social media. By doing so, businesses can enhance their productivity and efficiency, educators can improve student learning and engagement, healthcare providers can improve patient outcomes and access to care, and individuals can enhance their social connections while avoiding the negative impacts of addiction, misinformation, and decreased social interaction.

Overall, the responsible use of chatting apps can lead to significant benefits, and it is up to individuals to use these apps in a way that serves their needs while being mindful of the potential risks.

**7.FUTURE SCOPE**

There are several enhancements that can be made in the future for chatting apps. Here are some ideas:

Voice and Video Calling: Voice and video calling can be added to the chat app, making it possible for users to make audio and video calls to their contacts within the app. This would add more functionality to the app and make it a more versatile communication tool.

End-to-End Encryption: End-to-end encryption can be added to the chat app to ensure that all conversations are private and secure. This will protect users' personal information and make the app more trustworthy.

AI Chatbots: Chatbots powered by artificial intelligence can be integrated into the chat app to assist users with various tasks, such as booking appointments or making reservations. This will provide a more personalized experience for users and save them time.

Group Chat Features: Group chat features can be enhanced to allow users to create sub-groups within larger groups. This would enable users to have more focused discussions with specific individuals or on specific topics.

Integration with Other Apps: Chat apps can be integrated with other apps, such as calendars or task management tools, making it easier for users to manage their schedules and tasks within the app.

Emojis and Stickers: More emojis and stickers can be added to the app to enhance the user experience and make conversations more fun and engaging.

Customization Options: Customization options can be added to the app, allowing users to customize the app's interface to their liking. This would give users a sense of ownership and make the app more personalized.

Better Search Functionality: The search functionality can be improved to make it easier for users to find past conversations, contacts, or messages.

Integration with Wearable Devices: Chat apps can be integrated with wearable devices, such as smartwatches, making it possible for users to send and receive messages without having to take out their phones.

Multi-Language Support: Multi-language support can be added to the app, making it more accessible to users who speak different languages. This would expand the app's user base and increase its global reach.

**8.APPENDIX**

**A. Source code**

**AndroidManiFest.xml**

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.risbin.flashchat"**>  
  
 <**uses-permission android:name="android.permission.INTERNET"**/>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.FlashChat"**>  
 <**activity  
 android:name="com.risbin.flashchat.MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.FlashChat.NoActionBar"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"**/>  
 </**intent-filter**>  
 </**activity**>  
 </**application**>  
  
</**manifest**>

**MainActivity.kt file**

**package** com.risbin.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()  
 **}** }  
}

**NavComposeApp.kt file**

**package** com.risbin.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.risbin.flashchat.nav.Action  
**import** com.risbin.flashchat.nav.Destination.*AuthenticationOption***import** com.risbin.flashchat.nav.Destination.*Home***import** com.risbin.flashchat.nav.Destination.*Login***import** com.risbin.flashchat.nav.Destination.*Register***import** com.risbin.flashchat.ui.theme.FlashChatTheme  
**import** com.risbin.flashchat.view.AuthenticationView  
**import** com.risbin.flashchat.view.home.HomeView  
**import** com.risbin.flashchat.view.login.LoginView  
**import** com.risbin.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()  
 **}  
 }  
 }**}

**Constants.kt file**

**package** com.risbin.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 file**

**package** com.risbin.flashchat.nav  
  
**import** androidx.navigation.NavHostController  
**import** com.risbin.flashchat.nav.Destination.*Home***import** com.risbin.flashchat.nav.Destination.*Login***import** com.risbin.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() **}**}

**AuthenticationOption.kt file**

**package** com.risbin.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.material.\*  
**import** androidx.compose.runtime.Composable  
**import** androidx.compose.ui.Alignment  
**import** androidx.compose.ui.Modifier  
**import** androidx.compose.ui.graphics.Color  
**import** com.risbin.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 file**

**package** com.risbin.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  
  
@Composable  
**fun** Title(title: String) {  
 Text(  
 text = title,  
 fontSize = 30.sp,  
 fontWeight = FontWeight.Bold,  
 modifier = Modifier.fillMaxHeight(0.5f)  
 )  
}  
  
@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 file**

**package** com.risbin.flashchat.view.home  
  
**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.text.input.KeyboardType  
**import** androidx.compose.ui.unit.dp  
**import** androidx.lifecycle.viewmodel.compose.viewModel  
**import** com.risbin.flashchat.Constants  
**import** com.risbin.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,  
 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"** )  
 **}  
 }** )  
 **}**}

**HomeViewModel.kt file**

**package** com.risbin.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.risbin.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  
   
 fun** updateMessage(message: String) {  
 **\_message**.*value* = 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* = **""  
 }** }  
 }  
  
   
 **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)  
 **}** }  
  
   
 **private fun** updateMessages(list: MutableList<Map<String, Any>>) {  
 **\_messages**.*value* = list.*asReversed*()  
 }  
}

**Login.kt file**

**package** com.risbin.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.risbin.flashchat.view.Appbar  
**import** com.risbin.flashchat.view.Buttons  
**import** com.risbin.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 = 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 file**

**package** com.risbin.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**

**fun** updateEmail(newEmail: String) {  
 **\_email**.*value* = newEmail  
 }  
  
   
 **fun** updatePassword(newPassword: String) {  
 **\_password**.*value* = newPassword  
 }  
  
  
 **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 file**

**package** com.risbin.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.risbin.flashchat.view.Appbar  
**import** com.risbin.flashchat.view.Buttons  
**import** com.risbin.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** )  
 **}  
 }**}

**RegisterViewModel.kt file**

**package** com.risbin.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  
  
   
 fun** updateEmail(newEmail: String) {  
 **\_email**.*value* = newEmail  
 }  
  
  
 **fun** updatePassword(newPassword: String) {  
 **\_password**.*value* = newPassword  
 }  
  
  
 **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  
 }** }  
 }  
}