# Android Application Development

## Chat Connect-A real time chat connect and Communiction App

TEAM ID - NM2024TMID05713


Submitted By

**Mirdulaa S.R** (Team leader) - F0B5077A3E9499CAB434211D9D0B0838

**Priyadharshini K**(Team Member)-BAAFEB2EEB40321B6944DAA0DB4DB777

**Pragadeesh S**(Team Member)- 289DE1395F739A89EED4C958D1D74A8A

**Indhumathi D**(Team Member)- A8DF847A6284C32A876452963A4DC6C1


SEMESTER-V

B.E COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR-2024-2025



DEPARTMENT OF COMPUTER SCIENCE ENGINEERIING

ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE

COIMBATORE-641046


NOVEMBER-2024

## 1. PROJECT OVERVIEW

**ProjectName:**ChatConnect

**Objective:**

Chat Connect is a simple chat application built using Android's **Compose UI toolkit**. The app allows users to register, log in, and engage in a messaging experience with features such as emoji support, translation, handwriting recognition, and customizable keyboard functionality. This app leverages **Firebase** for backend support, ensuring real-time communication and data storage.

**Key Features:**

- **Registration & Login**: Users can register and log in securely using Firebase Authentication.
- **Messaging**: Users can send and receive text messages in real-time.
- **Emojis**: A rich selection of emojis for more interactive communication.
- **Translation**: Users can translate messages to their preferred language.
- **Handwriting Recognition**: A feature that converts handwritten notes to text.
- **Customizable Keyboard**: Users can modify the appearance of their keyboard.

**Technologies Used:**

- **Android Studio** (IDE)
- **Jetpack Compose** (UI Toolkit)
- **Firebase Authentication** (For login and user registration)
- **Firebase Realtime Database** (For real-time message storage)
- **Google Translate API** (For translation)
- **Firebase ML Kit** (For handwriting recognition)

## 2. OBJECTIVES

**Business/Functional Goals:**

- Provide users with a seamless chat experience with multiple features integrated into one app.
- Ensure the app is user-friendly and responsive across devices.
- Implement real-time communication and data syncing via Firebase.

**Specific Outcomes:**

- A fully functional app that allows user registration, login, and interaction through a real-time chat interface.
- Integration with Firebase for user management and message storage.
- Emoji and translation features to enhance the user experience.
- Handwriting-to-text functionality for users who prefer drawing over typing.
- A customizable keyboard to make the chat experience more engaging and personalized.

## 3. KEY FEATURES AND CONCEPTS UTILIZED

1. **Jetpack Compose UI Toolkit:**
    - Composable such as Text Field, Button, Column, Row, and Scaffold to create a responsive and interactive UI.
    - State management in Compose using remember, mutable State Of, and Launched Effect.
2. **Firebase Authentication:**
    - Firebase Authentication handles user sign-up and login functionalities. It securely manages user credentials and sessions.
    - Using **Firebase Auth** to implement email/password-based authentication for registration and login.
3. **Firebase Realtime Database:**
    - Storing and retrieving messages in real-time using Firebase Realtime Database. This ensures messages are sent and received instantly.

- **Firebase Database SDK** used to read/write data (messages, user details, etc.).

4. **Emoji Support:**
    - Integrating emoji functionality into the text input field so users can easily add emojis to their messages.
5. **Translation (Google Translate API):**
    - Messages can be translated into the user's preferred language via integration with the Google Translate API. This allows users from different regions to communicate more effectively.
6. **Handwriting Recognition (Firebase ML Kit):**
    - Integrating Firebase ML Kit for handwriting recognition allows users to draw messages on the screen, which are then converted into text.

7**.Keyboard Customization:**

- Allowing users to customize their chat keyboard by changing themes, colors, layouts for a personalized experience.

## 4. DETAILED STEPS TO SOLUTION DESIGN

### 1. Data Model Design:

- **User Data:** Each user has an account with unique identifiers (UID), email, and password stored via Firebase Authentication.
- **Messages:** Each message includes the sender's UID, the message content, timestamp, and the message's translated version if applicable.

### 2. User Interface Design:

- **Login & Registration Pages:**
    - Composable like Text Field, Button, and Column are used to create a login form and registration form.
    - User authentication is handled via Firebase Authentication.
- **Main Chat Screen:**

- **Message List:** A vertically scrolling Lazy Column displays messages in real-time.
- **Text Input:** A Text Field for typing messages, with an emoji button and a microphone for handwriting recognition.
- **Send Button:** A button to send the message to Firebase.
- **Translation and Handwriting Buttons:** Icons to trigger message translation or handwriting mode.

**3. Business Logic:**

- **Message Sending:** Once a message is entered and the send button is clicked, it is stored in Firebase Realtime Database under a unique chat room or user group.
- **Real-time Updates:** Messages are fetched in real-time from Firebase and displayed in the UI.
- **Emoji Support:** A custom emoji keyboard is implemented, allowing users to easily add emojis to their messages.
- **Handwriting Recognition:** The app uses Firebase ML Kit to convert hand-drawn characters to text in real-time.

**4. Firebase Integration:**

- **Firebase Authentication:** Handles the login and registration of users.
- **Firebase Realtime Database:** Used for storing and retrieving messages, ensuring that all users in the chat can see real-time updates.

**5. TESTING AND VALIDATION**

**Unit Testing:**

- Test the Firebase Authentication flow, ensuring users can successfully register, log in, and log out.

- Test the message sending and receiving functionality to ensure messages are properly written to and retrieved from Firebase.

**UI Testing:**

- Test the user interface for responsiveness, especially for devices with different screen sizes.
- Ensure the emoji and handwriting input functionalities are working correctly and integrate seamlessly with the message input.

**Functional Testing:**

- Validate the translation feature by ensuring it works across different languages.
- Test the handwriting-to-text feature, making sure it recognizes different handwriting styles accurately.

**Integration Testing:**

- Test the integration of Firebase with the app, ensuring data is stored and retrieved without errors.
- Ensure real-time message syncing works smoothly between users.

## 6. KEY SCENARIOS ADDRESSED BY THE APP

- **User Registration and Login:**
    - Users can create accounts and log into the app securely.
    - The app will handle errors such as invalid credentials and provide appropriate feedback.
- **Real-time Messaging:**
    - Once logged in, users can send and receive messages in real-time. Messages are stored in Firebase and synchronized across users.
- **Emoji Integration:**
    - Users can easily insert emojis into messages using a custom emoji keyboard.
- **Message Translation:**

- Users can translate messages into different languages, enhancing cross-lingual communication.

- **Handwriting-to-Text:**
  - Users can draw messages, which are converted into text using Firebase ML Kit.

- **Keyboard Customization:**
  - Users can modify the appearance of their keyboard, choosing themes and layouts to personalize their chat experience.

## 7. CONCLUSION

**Summary of Achievements:**

- **Successful Integration with Firebase:** The app seamlessly integrates Firebase Authentication and Firebase Realtime Database to enable secure login, registration, and real-time messaging.
- **User-friendly UI:** The app's design uses Jetpack Compose to create a modern, responsive, and user-friendly interface.
- **Enhanced Communication Features:** Features like emojis, message translation, and handwriting recognition provide users with a rich and interactive chat experience.
- **Customizable Keyboard:** Users are able to personalize their keyboard, making the chat experience more enjoyable.

This project provides a solid foundation for building more complex chat applications, with the flexibility to add more features, such as voice messaging or video chat, in the future. The integration of modern Android development tools, such as Jetpack Compose and Firebase, enables developers to create scalable, real-time apps efficiently.

**CODE IMPLEMENTATION:**

## MainActivity.kt

```
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**

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

**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.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() }
}
```

**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
val login: () -> Unit = { navController.navigate(Login) }
   val register: () -> Unit = { navController.navigate(Register) }
   val navigateBack: () -> Unit = { navController.popBackStack() }
}


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

```
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

   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)
            }
_loading.value = true
            auth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener {
                    if (it.isSuccessful) {
                        home()
                    }
                    _loading.value = false
                }
        }
    }
}
```

**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.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 = 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.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 = 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
        )
    }
  }
}
```

**OUTPUT**

## 7. CONCLUSION

**Summary of Achievements:**

- **Successful Integration with Firebase:** The app seamlessly integrates Firebase Authentication and Firebase Realtime Database to enable secure login, registration, and real-time messaging.
- **User-friendly UI:** The app's design uses Jetpack Compose to create a modern, responsive, and user-friendly interface.
- **Enhanced Communication Features:** Features like emojis, message translation, and handwriting recognition provide users with a rich and interactive chat experience.
- **Customizable Keyboard:** Users are able to personalize their keyboard, making the chat experience more enjoyable.

This project provides a solid foundation for building more complex chat applications, with the flexibility to add more features, such as voice messaging or video chat, in the future. The integration of modern Android development tools, such as Jetpack Compose and Firebase, enables developers to create scalable, real-time apps efficiently.