ChatConnect - A Real-Time Chat and Communication App

PROJECT REPORT

Submitted by

Jenisha S (20203111506131)

Jersha Ujina Thaddeus (20203111506132)

Jeshel J P (20203111506133)

Jincy R (20203111506134)

Contents

Preface

1.INTRODUCTION

- **Overview**
- **Purpose**

2.PROBLEM DEFINITION AND DESIGN THINKING

2.1 Empathy Map

2.2 Ideation and Brainstorming Map

3. RESULT

4. ADVANTAGES AND DISADVANTAGES

5. APPLICATION

6.CONCLUSION

7.FUTURE SCOPE

8.APPENDIX

SOURCE CODE

1.Introduction:

1.1 Overview:

A messaging application is a mobile-phone-based software program that allows users to send and receive information using their phone's internet connection. Messaging apps can transmit or receive a much wider range of data types than Short Message Service (SMS) or Multimedia Messaging Service (MMS).Modern instant messaging and SMS both began their march to prominence in the early and mid-1990s. The difference between the two is subtle: SMS (the acronym for "short service message") allows mobile phone users to send each other text messages without an Internet connection, whereas instant messaging enables similar functionality via the web.

The first SMS message was sent over the Vodafone GSM network in the United Kingdom on December 3, 1992, with the words "Merry Christmas." Israeli firm Mirabilis released the first widely used online messenger, ICQ (short for "I Seek You"), in 1996.As Wi-Fi and high-speed mobile networks hit critical mass in many markets, chat apps quickly became multimedia hubs where users could easily share videos, photos, stickers, games, articles, live streams, and more. Between 2013 and 2014, many messengers turned their attention to monetizing their massive audiences, and in doing so introduced tools for publishers and brands.

Firebase:

Firebase is a Backend-as-a-Service (Baas). It is a Google-backed app development platform that allows developers to build iOS, Android, and web apps. It provides tools for tracking analytics, reporting and fixing app crashes, and creating marketing and product experiments. These help developers develop quality apps, grow their user base, and make a profit. We will be using two of their tools: Firebase Authentication and Cloud Fire store.

Firebase Authentication:

Firebase Authentication (SDK) is a Firebase tool that supports different authentication methods like passwords, phone numbers, Google, Facebook, Twitter, GitHub, and more. In this app, we will be using the Google sign-in Authentication.

Cloud Fire store:

Cloud Fire store is a cloud-based NoSQL database server for storing and syncing data. It stores data in documents as key-value pairs, and the documents are organized into collections.The documents can also have sub-collections, allowing you to nest collections within collections. The data is also synchronized automatically among all devices listening for them.Now that you have an idea of how Firebase and store work, let’s build our app.

Cloud Storage for Firebase:

Cloud Storage for Firebase is built for app developers who need to store and serve user-generated content, such as photos or videos. Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality.You can use our SDKs to store images, audio, video, or other user generated content.

1.2 Purpose:-

Chatting app allows you to communicate with your customers in web chat rooms. It enables you to send and receive messages. Chatting apps make it easier, simpler, and faster to connect with everyone and it is also easy to use. There are many types of chatting apps and every one has its own format, design, and functions.

A Chatconnect application is introduced that will enable real-time text messaging between two users on a network, as well as the transfer of assets including photographs, audio, and videos. To handle the backend of the communication operation, the Android operating system is used and Google Firebase, showcasing the numerous functionalities of both the operating system and the service. It also showcases the use of Compose's declarative UI and state management capabilities. It also includes examples of how to handle input and navigation using

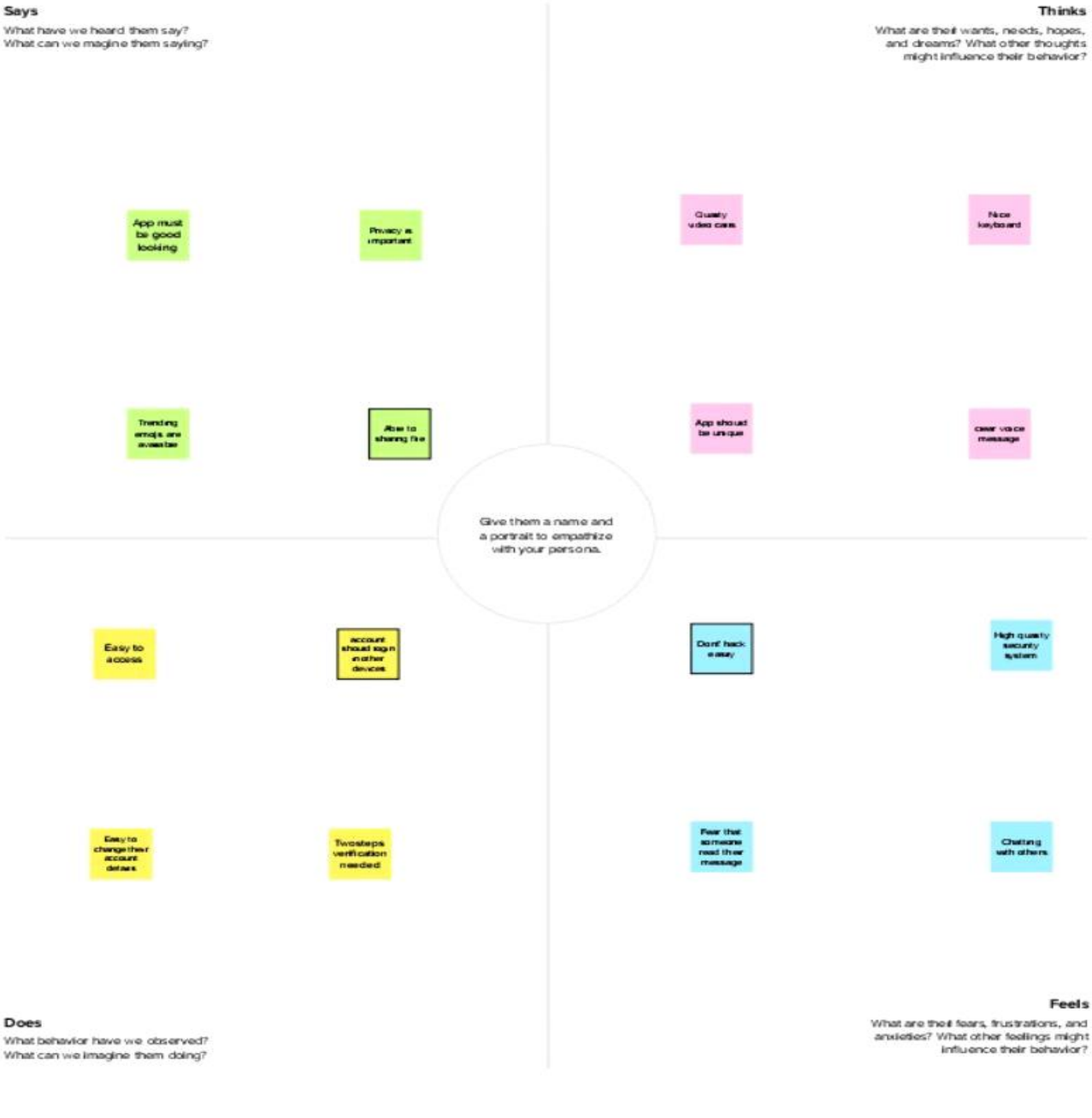
composable functions and how to use data from a firebase to populate the UI.

2.PROBLEM DEFINITION AND DESIGN THINKING

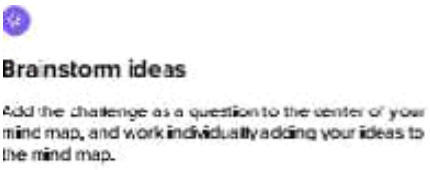
2.1 Empathy Map

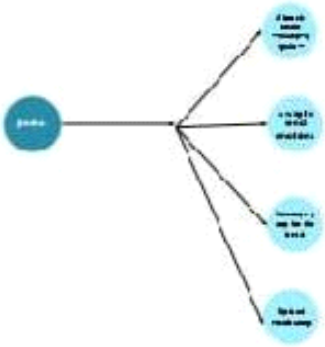
Build empathy

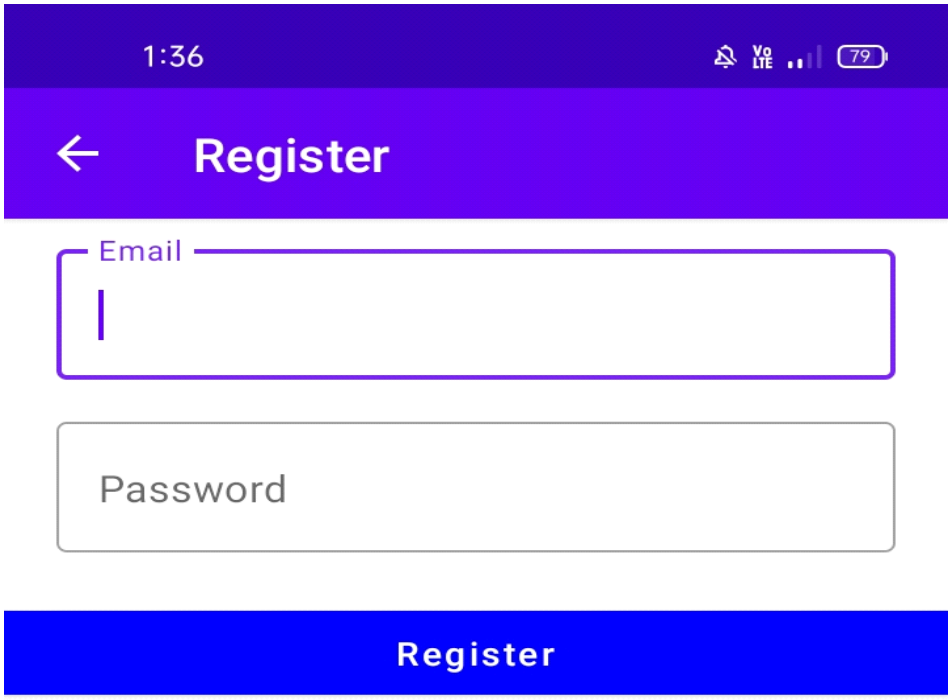
The information you add here should be representative of the observations and research you've done about your users.



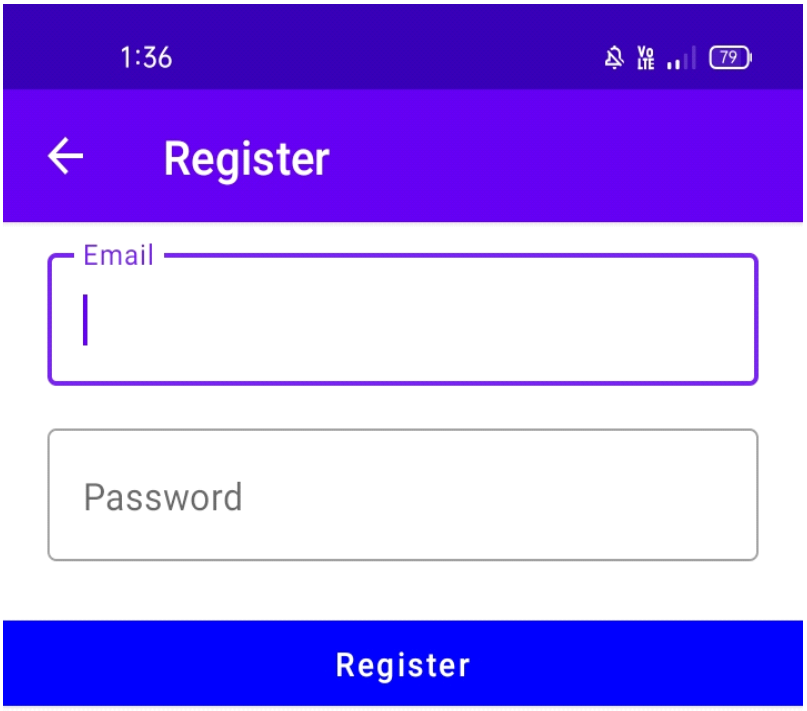
2.2 IDEATION AND BRAINSTORMING MAP

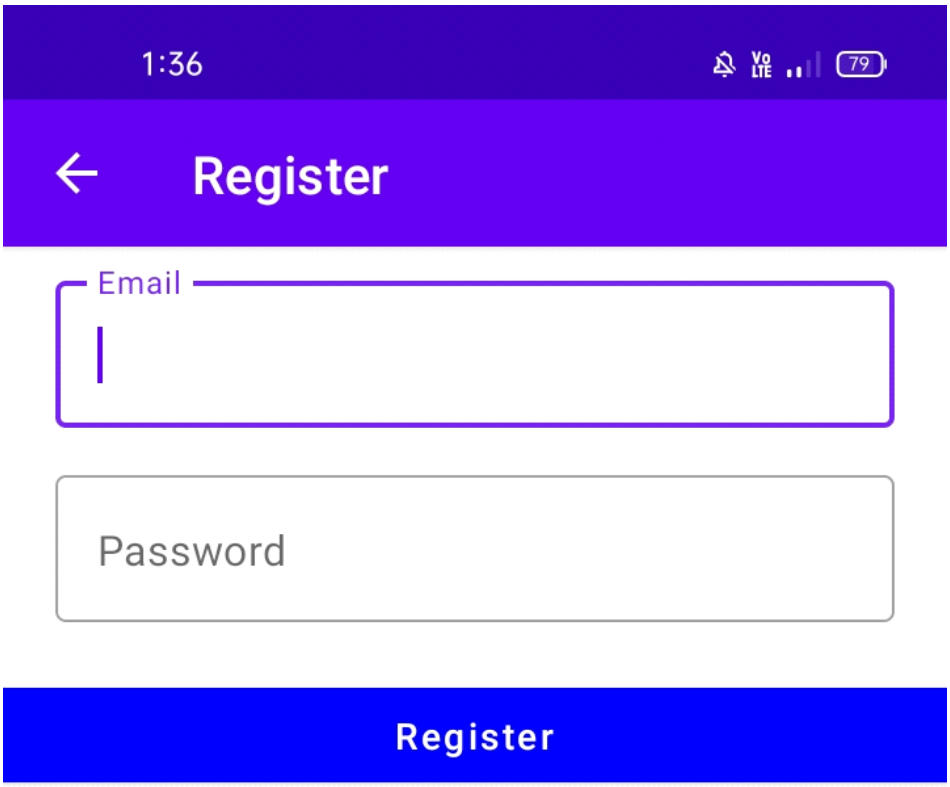


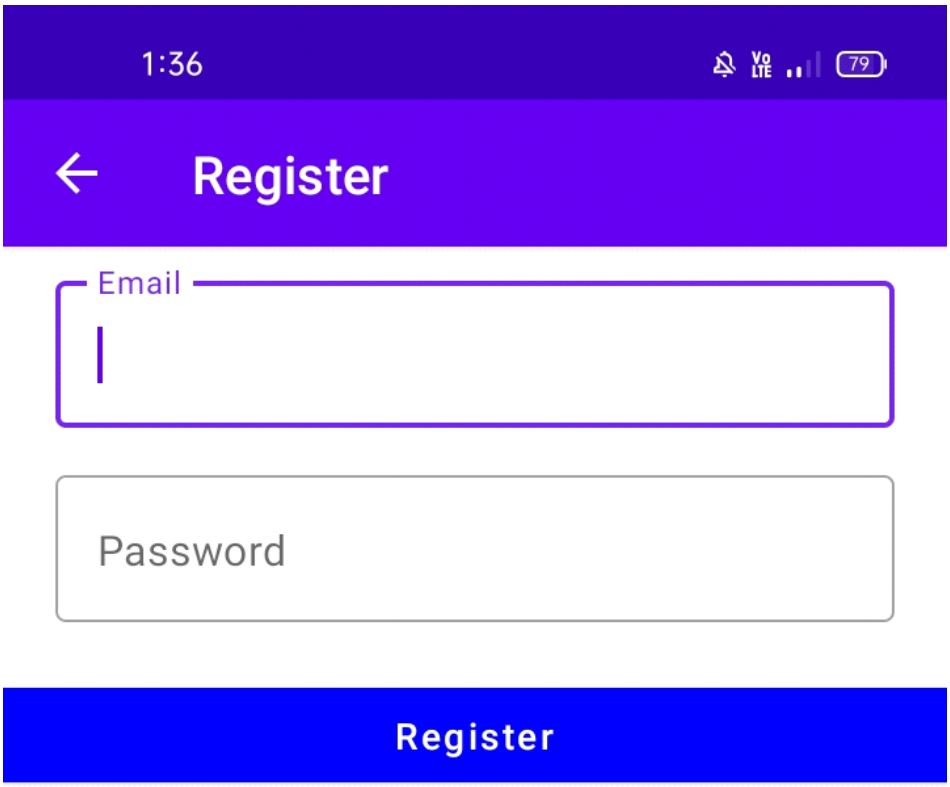




3.RESULT


1:36
Register
Email
Password
Register


1:36
79
Register
Email
Password
Register


1:36
79
Register
Email
Password
Register

4.ADVANTAGES AND DISADVANTAGES

4.1 ADVANTAGES

A real-time chat application is a popular feature in many mobile applications, including Android apps. Google Firebase is a widely used platform for developing chat applications on the Android platform due to its real-time database and other features. Some of the advantages are:

Real-time updates:

Firebase provides real-time updates, allowing users to receive messages instantly as they are sent. This feature is essential for a chat application where users need to receive messages quickly and efficiently.

Scalability:

Firebase is designed to handle large amounts of data, making it suitable for chat applications with a high volume of users. As your chat application grows, Firebase can easily scale up to accommodate the increased traffic, ensuring that users continue to receive fast and responsive service.

Offline support:

Firebase provides offline support, allowing users to send and receive messages even when they are not connected to the internet. This feature ensures that users can stay connected to the

chat application even in low connectivity areas, such as subways or remote areas.

Cross-platform compatibility:

Firebase supports multiple platforms, including Android, iOS, and web, making it easy to create a chat application that can be accessed from any device. This feature is particularly useful if you plan to develop your chat application for multiple platforms, ensuring that users can stay connected regardless of the device they are using.

Security:

Firebase provides secure user authentication and data storage, ensuring that user data is protected. This feature is essential for any chat application, as user privacy and security are critical.

Analytics and Monitoring:

Firebase provides robust analytics and monitoring tools that help developers understand user behavior and performance issues. This feature is particularly useful for chat applications, as it allows developers to track engagement levels and ensure that the application is performing optimally.

4.2 DISADVANTAGES

Google Firebase is a popular choice for developing real-time chat applications on the Android platform. It offers several

advantages such as real-time updates, scalability, offline support, cross-platform compatibility, and security. However, it also

some disadvantages such as a steep learning curve, cost, limited features, and vendor lock-in.

Learning curve:

Firebase can have a steep learning curve for developers who are new to the platform. If not familiar with Firebase, it may need to spend some time learning how to use it effectively. Additionally, Firebase has a unique data model, which may require some adjustment if you are used to working with traditional SQL databases.

Cost:

While Firebase has a free plan, it can become costly as the usage and the number of users grow. Firebase charges based on the amount of data stored, the number of users, and the amount of data transferred. As you scale your chat application, you may need to upgrade to a paid plan to meet your needs.

Limited features:

Firebase has a limited set of features compared to other chat application platforms, which may be a disadvantage for more complex applications. For example, Firebase's real-time database is designed for simple data structures and may not be suitable for more complex chat applications.

Vendor lock-in:

Developing an application with Firebase can create a dependency on the platform, which can be difficult to move away from if needed. If you ever decide to move your application to a different platform, it may require significant effort and resources.

Performance issues:

While Firebase is designed to handle large amounts of data, it may not be suitable for extremely high-traffic chat applications. If your application experiences high levels of traffic, you may need to consider using a different platform that can handle the load more efficiently.

5. APPLICATION

Messaging has become a part of our everyday lives in part due to its convenience for real-time chat communication and simple-to-use functionality. For instance, an iOS or text message on an iPhone or Android device from a friend, an email from a co-worker on Microsoft or Gmail, a team chat in a Slack or Microsoft Teams workspace, or even instant messaging through social media. These messaging and real-time chat applications play an important role in how the world interacts today, due to their immediacy and vast capabilities.

The benefits of a real-time chat application

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. This also keeps users conversing on your platform instead of looking elsewhere for a messaging solution. Whether it's private chat, group chat, or large-scale chat, adding personalized chat features to your app can help ensure that your users have a memorable experience.

In this chat application guide, we'll explore:

• How custom in-app chat features enable user engagement

• The long-term factors to consider when building a real-time chat Application

• How engaging messaging apps lead to an improved user experience

By the end of this guide, you'll understand the benefits of implementing personalized, real-time chat app features to increase user engagement and retention on your platform.

What to consider when building real-time chat Applications

More people are embracing virtual experiences as a way to connect with one another. From gamer group chat messaging in a live chat to e-learning and team communication using chat

rooms and file sharing among co-workers, Online chat applications have grown largely in popularity because of their Ability to retain the feel of a real-time conversation, virtually. However, when Thinking ahead to how you want to build your web app, a vital thing to Consider is your customer's experience. What are the essential features and Functionalities that are needed to make an engaging real-time chat app that Will lead to user retention? Let's dive in.

Real time chat needs engaging messaging

Features

A digital chat app that's composed of real-time messaging features enables Users to have an authentic and interactive experience.

Features like message reactions, stickers, emojis, GIFs, and voice calls and Video chat, provide a way to engage your users directly on your app instead of External platforms–creating a more connected experience.

Other functionalities like identifying active users, push notifications, and Message history—to name a few—also add to that immediacy by automatically Detecting the presence of users in a real-time chat application.

Scalability of your real-time chat app

When building a real-time chat application, another key factor to consider is Concurrency. Whether a private chat, group chat, or large-scale chat Experience, being able to build your chat

app without worrying about user Fluctuations and concurrency limits on your platform is crucial.

6. Conclusion

Chat support is a form of synchronous messaging. Agent and Client must be present at the same time. Typically, this Communication channel is available on websites in the form of a Popup dialog. Thanks to this, the agent can quickly provide Information on what is bothering the customer. There are a few rules That you should keep in mind as the basic chat rules. The first is to Maintain correct grammar and spelling. Errors make the customer Service team less competent in the eyes of the customer. Another Rule is the fastest possible response, so it's important that agents Write quickly when they have a conversation. Another important rule Is not to use acronyms and abbreviations. The use of such forms is Not professional. It is similar with writing short "yes" and "no" Answers. Always try to use complete sentences. Ready-made Answers support the work of the customer service team, but don't Send them out thoughtlessly. Agents should always personalize the Response. It's nice when we address the client by name. The Customer then feels that the service team is addressing him and is Not one of many.

7. 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 Authorization service.
- Creating Database and maintaining users.
- Increasing the effectiveness of the application by providing Voice Chat.
- Extending it to Web Support.

As for the Chat connect application, it has numerous upgrade paths from implementing call functionality or any other enhanced future works to the flexible nature of the application and the application will continue to upgrade as long as the technology upgrades. So, the application is extremely future-proof.

8. Appendix :

Source code:

Main activity.kt file

```
package com.project.pradyotprakash.flashchat

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.Firebase
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      FirebaseApp.initializeApp(this)
```

```
        setContent {
            NavComposeApp()
        }
    }
}
```

Creating NavComposeApp.kt file:

Package com.project.pradyotprakash.flashchat

Import androidx.compose.runtime.Composable

Import androidx.compose.runtime.remember

Import androidx.navigation.compose.NavHost

Import androidx.navigation.compose.composable

Import androidx.navigation.compose.rememberNavController

Import com.google.firebase.auth.FirebaseAuth

Import com.project.pradyotprakash.flashchat.nav.Action

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

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

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

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

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

Import com.project.pradyotprakash.flashchat.view.AuthenticationView

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

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

Import com.project.pradyotprakash.flashchat.view.register.RegisterView

```
@Composable
Fun NavComposeApp() {
    Val navController = rememberNavController()
    Val actions = remember(navController) { Action(navController) }
    FlashChatTheme {
        NavHost(
            navController = navController,
            startDestination =
            if (FirebaseAuth.getInstance().currentUser != null)
                Home
            Else
                AuthenticationOption
        ) {
            Compostable(AuthenticationOption) {
                AuthenticationView(
                    Register = actions.register,
                    Login = actions.login
                )
            }
            Composable(Register) {
                RegisterView(
                    Home = actions.home,
                    Back = actions.navigateBack
                )
            }
```

```
        Composable(Login) {
            LoginView(
                Home = actions.home,
                Back = actions.navigateBack
            )
        }
        Composable(Home) {
            HomeView()
        }
    }
  }
}
```

Creating Constant object:

```
package com.project.pradyotprakash.flashchat

object Constants {
    const val TAG = "flash-chat"

    const val MESSAGES = "messages"
    const val MESSAGE = "message"
    const val SENT_BY = "sent_by"
    const val SENT_ON = "sent_on"
    const val IS_CURRENT_USER = "is_current_user"
}
```

Creating Navigation.kt in Nav package :

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

import androidx.navigation.NavHostController
import com.project.pradyotprakash.flashchat.nav.Destination.Home
import com.project.pradyotprakash.flashchat.nav.Destination.Login
import com.project.pradyotprakash.flashchat.nav.Destination.Register


object Destination {
    const val AuthenticationOption = "authenticationOption"
    const val Register = "register"
    const val Login = "login"
    const val Home = "home"
}

class Action(navController: NavHostController) {
    val home: () -> Unit = {
        navController.navigate(Home) {
            popUpTo(Login) {
                inclusive = true
            }
            popUpTo(Register) {
                inclusive = true
            }
        }
    }
    val login: () -> Unit = { navController.navigate(Login) }
    val register: () -> Unit = { navController.navigate(Register) }
    val navigateBack: () -> Unit = { navController.popBackStack() }
}
```

Creating AuthenticationOption.kt file in view package:

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

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import com.project.pradyotprakash.flashchat.ui.theme.FlashChatTheme

@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {
    FlashChatTheme {
        // A surface container using the 'background' color from the theme
        Surface(color = MaterialTheme.colors.background) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement. Bottom
            ) {
                Title(title = "⚡ Chat Connect")
                Buttons(title = "Register", onClick = register, backgroundColor =
Color.Blue)
                Buttons(title = "Login", onClick = login, backgroundColor =
Color.Magenta)
            }
        }
    }
}
```

Creating widgets.kt file in view package :

```
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.sp
import androidx.compose.ui.unit.sp
import com.project.pradyotprakash.flashchat.Constants

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

// Different set of buttons in this page
@Composable
fun Buttons(title: String, onClick: () -> Unit, backgroundColor: Color) {
    Button(
        onClick = onClick,
        colors = ButtonDefaults.buttonColors(
```

```
            backgroundColor = backgroundColor,
            contentColor = Color.White
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = RoundedCornerShape(0),
    ) {
        Text(
            text = title
        )
    }
}

@Composable
fun Appbar(title: String, action: () -> Unit) {
    TopAppBar(
        title = {
            Text(text = title)
        },
        navigationIcon = {
            IconButton(
                onClick = action
            ) {
                Icon(
                    imageVector = Icons.Filled.ArrowBack,
                    contentDescription = "Back button"
                )
            }
        }
    )
}

@Composable
fun TextFormField(value: String, onValueChange: (String) -> Unit, label: String,
keyboardType: KeyboardType, visualTransformation: VisualTransformation) {
    OutlinedTextField(
        value = value,
        onValueChange = onValueChange,
        label = {
            Text(
                label
```

```
            )
        },
        maxLines = 1,
        modifier = Modifier
            .padding(horizontal = 20.dp, vertical = 5.dp)
            .fillMaxWidth(),
        keyboardOptions = KeyboardOptions(
            keyboardType = keyboardType
        ),
        singleLine = true,
        visualTransformation = visualTransformation
    )
}

@Composable
fun SingleMessage(message: String, isCurrentUser: Boolean) {
    Card(
        shape = RoundedCornerShape(16.dp),
        backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else
Color.White
    ) {
        Text(
            text = message,
            textAlign =
            if (isCurrentUser)
                TextAlign.End
            else
                TextAlign.Start,
            modifier = Modifier.fillMaxWidth().padding(16.dp),
            color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White
        )
    }
}
```

Creating Home.kt file in home package in view package :

```
Package com.project.pradyotprakash.flashchat.view.home

Import androidx.compose.foundation.background
Import androidx.compose.foundation.layout.*
Import androidx.compose.foundation.lazy.LazyColumn
Import androidx.compose.foundation.lazy.items
Import androidx.compose.foundation.text.KeyboardOptions
Import androidx.compose.material.*
Import androidx.compose.material.icons.Icons
Import androidx.compose.material.icons.filled.Send
Import androidx.compose.runtime.Composable
Import androidx.compose.runtime.getValue
Import androidx.compose.runtime.livedata.observeAsState
Import androidx.compose.ui.Alignment
Import androidx.compose.ui.Modifier
Import androidx.compose.ui.graphics.Color
Import androidx.compose.ui.text.input.KeyboardType
Import androidx.compose.ui.unit.dp
Import androidx.lifecycle.viewmodel.compose.viewModel
Import com.project.pradyotprakash.flashchat.Constants
Import com.project.pradyotprakash.flashchat.view.SingleMessage

@Composable
Fun HomeView(
   homeViewModel: HomeViewModel = viewModel()
) {
   Val message: String by homeViewModel.message.observeAsState(initial = “”)
   Val messages: List<Map<String, Any>> by
homeViewModel.messages.observeAsState(
      Initial = emptyList<Map<String, Any>>().toMutableList()
   )

   Column(
      Modifier = Modifier.fillMaxSize(),
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.Bottom
   ) {
      LazyColumn(
         Modifier = Modifier
```

```
            .fillMaxWidth()
            .weight(weight = 0.85f, fill = true),
        contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
        verticalArrangement = Arrangement.spacedBy(4.dp),
        reverseLayout = true
    ) {
        Items(messages) { message ->
            Val isCurrentUser = message[Constants.IS_CURRENT_USER] as
Boolean

            SingleMessage(
                Message = message[Constants.MESSAGE].toString(),
                isCurrentUser = isCurrentUser
            )
        }
    }
    OutlinedTextField(
        Value = message,
        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”
                )
            }
        }
    )
  }
}
```

Creating HomeViewModel.kt file in home package in view package :

```
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().set(
                hashMapOf(
                    Constants.MESSAGE to message,
                    Constants.SENT_BY to Firebase.auth.currentUser?.uid,
                    Constants.SENT_ON to System.currentTimeMillis()
                )
            ).addOnSuccessListener {
                _message.value = ""
            }
        }
    }

    /**
     * Get the messages
     */
    Private fun getMessages() {
          Firebase.firestore.collection(Constants.MESSAGES)

          .orderBy(Constants.SENT_ON)

          .addSnapshotListener { value, e ->
```

```
            If (e != null) {
                Log.w(Constants.TAG, “Listen failed.”, e)
                return@addSnapshotListener
            }

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

            If (value != null) {
                For (doc in value) {
                    Val data = doc.data
                    Data[Constants.IS_CURRENT_USER] =
                        Firebase.auth.currentUser?.uid.toString() ==
data[Constants.SENT_BY].toString()

                    List.add(data)
                }
            }

            updateMessages(list)
        }
    }

    /**
     * Update the list after getting the details from firestore
     */
    Private fun updateMessages(list: MutableList<Map<String, Any>>) {
```

```
        _messages.value = list.asReversed()
    }
}
```

Creating login.kt file in login package in view package:

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

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

/**
 * 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
        )
    }
  }
}
```

Creating LoginViewModel.kt file in login package in view package :

```
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
                }
        }
    }
}
```

Creating Register.kt file in Register package in view package :

Package com.project.pradyotprakash.flashchat.view.register

Import androidx.compose.foundation.layout.*

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

/**
 * 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 = KeyboardType.Email,
                visualTransformation = VisualTransformation.None
            )
            TextFormField(
                Value = password,
                onValueChange = { registerViewModel.updatePassword(it) },
                label = “Password”,
                keyboardType = KeyboardType.Password,
                visualTransformation = PasswordVisualTransformation()
            )
            Spacer(modifier = Modifier.height(20.dp))
            Buttons(
                Title = “Register”,
                onClick = { registerViewModel.registerUser(home = home) },
                backgroundColor = Color.Blue
            )
        }
    }
}
```

Creating RegisterViewModel.kt file in Register package in view package :

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
            }
      }
   }
}
```