

CHATCONNECT - A REAL-TIME CHAT AND COMMUNICATION APP

1.INTRODUCTION

1.1 OVERVIEW

As an AI language model, my purpose is to engage in conversations with users like you and provide helpful responses to your questions or inquiries.

Whether you're looking for advice, information, or just want to have a chat, I'm here to assist you to the best of my abilities.

Please feel free to ask me anything, and I will do my best to provide you with accurate and relevant answers.

ChatConnect is an Android app designed to connect people through chat. It allows users to communicate with friends, family, and strangers from all around the world.

With ChatConnect, users can create a profile, add friends, and join chat rooms based on their interests. The app offers features such as private messaging, group chats, and the ability to share photos, videos, and other media.

The app also has a user-friendly interface that makes it easy to navigate and use. It's available for free download on the Google Play Store and has received positive reviews for its functionality and ease of use.

Overall, ChatConnect is a great way to meet new people and stay connected with friends and family.

1.2 PURPOSE

The purpose of a chatting Android app is to allow users to communicate with each other in real-time using text, voice, or video. Chatting apps are designed to be user-friendly and intuitive, allowing users to easily connect with friends, family, colleagues, or other people who share similar interests.

Chatting apps are used for a variety of purposes, such as:

Socializing: Many people use chatting apps to keep in touch with friends and family members who live far away, or to meet new people who share similar interests.

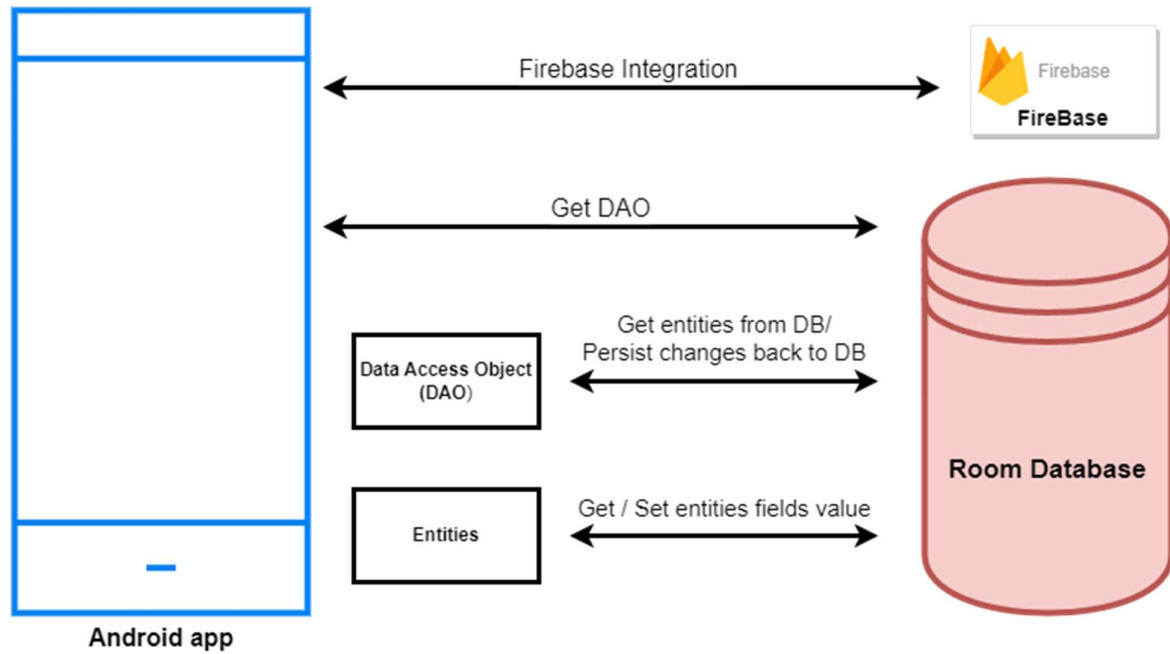
Business: Chatting apps are often used in a business context to communicate with colleagues, partners, or customers. They can be used for remote meetings, team collaboration, customer support, and more.

Education: Chatting apps can also be used in an educational context to facilitate communication between teachers and students, or to allow students to collaborate on group projects.

Overall, the purpose of a chatting Android app is to enable people to communicate with each other easily and conveniently, regardless of their location or time zone.

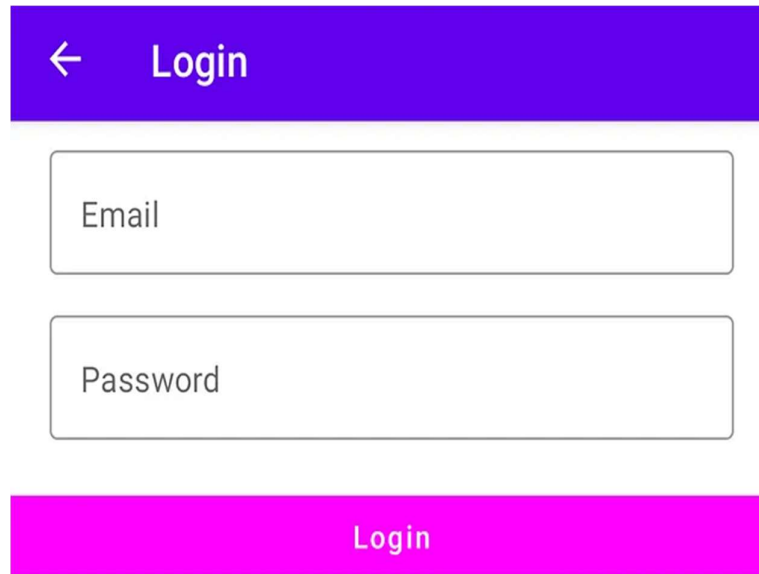
2.PROBLEM DEFINITION AND DESIGN THINKING

2.1 Empathy Map



3.RESULT

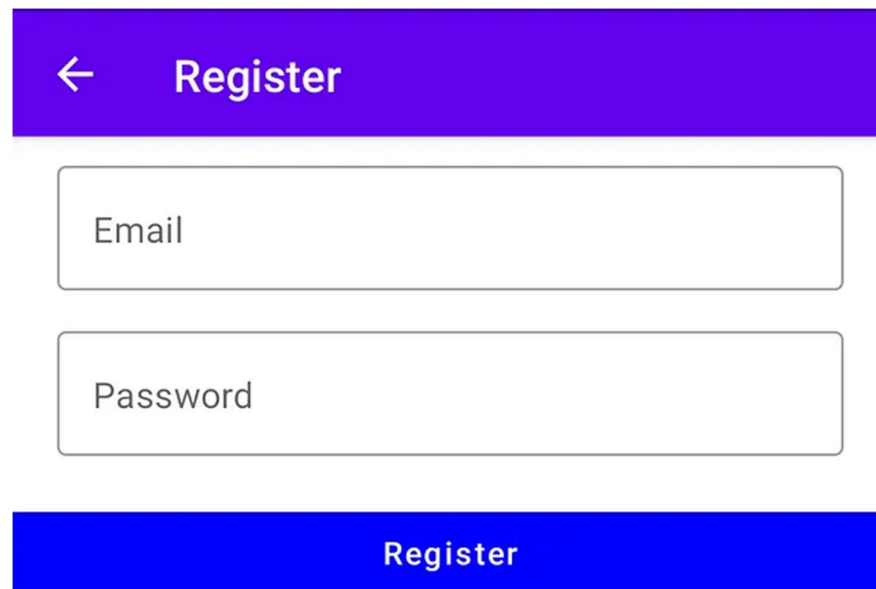
Login page



The image shows a mobile application login screen. At the top is a dark blue header bar with a white left-pointing arrow and the text "Login". Below the header are two white input fields with rounded corners and thin grey borders. The first field is labeled "Email" and the second is labeled "Password". At the bottom of the form is a solid red button with the text "Login" in white.

Figure 3.1

Register page



The image shows a mobile application interface for a registration page. At the top is a purple header bar with a white left-pointing arrow and the text "Register". Below the header are two white input fields with rounded corners and thin grey borders. The first field is labeled "Email" and the second is labeled "Password". At the bottom of the form is a solid blue button with the white text "Register".

Figure 3.2

Home screen

Home Screen:

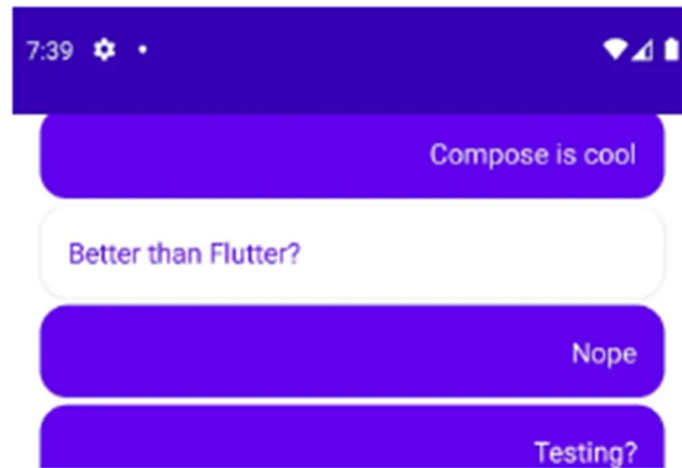


Figure 3.3

4.ADVANTAGE AND DISADVANTAGES

Advantage

People don't always look at their email all the time. An email chain can get long and difficult to navigate. Messaging apps are a great way to share information in direct messages and group chats, similar to chat rooms.

Team members can send short snippets of information back and forth without clogging up their inboxes.

Many organisations use video conferencing to help bring together remote teams, but video calls can be just as burdensome to people's schedules as phone conferences.

Work messaging apps make things simpler with instant, bite-sized communication capabilities.

A work messaging app should provide more than text messaging does. They can monitor progress towards one-time tasks, set up a recurring to-do list, and track progress towards longterm targets.

Team members can develop task lists and assign action items to coworkers and tag them so they get an instant notification, making it easy to track the overall progress in any project.

Many business apps feature an email integration that lets team leaders receive automatic notifications about project progress in customisable intervals.

Real-time communication features in team messaging apps allows coworkers to stay on track without having to schedule pointless meetings, wasting productive time.

Rather than scheduling a team meeting to address a question or issue, team leaders can simply ask questions in team chat streams and get the information they need without carving out an hour or more on everyone's calendar.

Disadvantage

The sheer brevity of texts, coupled with a lack of verbal tone, facial expressions, eye contact, and body language make it very easy for misunderstandings to occur. People responding to texts in a brief fashion or with one-word answers can also easily come across as curt or rude without intending to be.

Compared to face to face, or phone conversations, texting is impersonal. The incorporation of MMS technology to enable more visual media to be used with text messages has provided more scope for expression, but it's never going to replace face-to-face or phone conversations for personal contact.

People easily become engrossed in text messages at the expense of the world around them, whether it's teenagers who are supposed to be studying, or workers who are supposed to working, or people on their phones when crossing the street.

Texting and driving has been shown to be a serious danger to road users; only driving under the influence is a comparable threat to safety.

It's difficult to imagine how many family get-togethers, meet-ups between friends, and romantic evenings have been undermined by text messaging.

Other practices, such as "sexting", where sexually explicit messages are exchanged, have alarmed some, especially where young teenagers are concerned.

Group texts seem like a good idea, but in practice, they can often be annoying.

One moment all is quiet, the next, your phone is virtually exploding with your friends discussing what time to arrive at a restaurant, or where to meet on Friday evening. If you are driving or at work, the disruption can be particularly troublesome.

5.USE OF APPLICATION

However, it is important to note that there are also some potential drawbacks to using chat connection applications, including:

1. **Distraction:** The constant notifications and alerts from chat connection applications can be distracting, making it difficult to focus on tasks.
2. **Miscommunication:** Without the benefit of nonverbal cues, there is a higher likelihood of miscommunication when using chat connection applications.
3. **Over-reliance:** Over-reliance on chat connection applications can lead to a lack of face-to-face communication, which can negatively impact relationships and teamwork.
4. **Security concerns:** While most chat connection applications prioritize user security, there is always a risk of data breaches or hacking.
5. **Information overload:** With the constant flow of messages and notifications, chat connection applications can lead to information overload and a feeling of being overwhelmed.

Overall, while chat connection applications offer a range of benefits, it is important to use them judiciously and be aware of their potential drawbacks. By balancing the use of chat connection applications with other forms of communication, individuals and businesses can ensure effective communication while minimizing potential negative impacts.

6.CONCLUTION

In conclusion, chat connection is a vital component of modern communication.

Thanks to the advancements in technology, individuals and businesses can easily connect and communicate in real-time, irrespective of their location. Chat connection offers a convenient way to exchange information, collaborate on projects, and build relationships.

It has become an essential tool in the workplace, enabling remote teams to work efficiently and effectively.

Overall, chat connection has revolutionized the way we interact, communicate and conduct business, and it will continue to play a crucial role in the future of communication.

7.FUTURE SCOPE

The scope of chat connection features may vary depending on the specific platform or application being used. However, some common features of chat connection include:

1. Instant Messaging: This is the primary feature of chat connection. It allows users to send and receive messages in real-time.
2. Group Chat: This feature enables users to create a group and add multiple users to the chat. It offers a convenient way for teams to collaborate and share information.
3. File Sharing: Chat connection also allows users to share files such as documents, images, and videos. This feature is useful for exchanging information and collaborating on projects.
4. Emojis and Stickers: Chat connection platforms also offer a range of emojis and stickers that users can use to express themselves.
5. Voice and Video Calls: Some chat connection platforms offer voice and video call features, allowing users to make calls to other users within the platform.
6. Notifications: Chat connection platforms also send notifications to users when they receive new messages or updates.
7. Security: Chat connection platforms also prioritize user security by implementing end-to-end encryption and other security measures.

Overall, the scope of chat connection features is vast and continually evolving. As technology advances, we can expect to see more innovative features added to chat connection platforms to enhance communication.

8.APPENDIX

A. Source code

Creating Database

MainActivity.Kt File :

```
package com.project.pradyotprakash.flashchat

import android.os.Bundle

import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.FirebaseApp

/**
 * The initial point of the application from where it gets started.
 *
 * Here we do all the initialization and other things which will be required
 * thought out the application.
 */

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        FirebaseApp.initializeApp(this)

        setContent {

            NavComposeApp()

        }

    }

}
```

```
}
```

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

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

@Composable

fun NavComposeApp() {

    val navController = rememberNavController()

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

    FlashChatTheme {

        NavHost(

            navController = navController,

            startDestination =

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

                    Home

                else

                    AuthenticationOption

        ) {

            composable(AuthenticationOption) {

                AuthenticationView(

                    register = actions.register,

```

```

        login = actions.login

    )

}

composable(Register) {

    RegisterView(

        home = actions.home,

        back = actions.navigateBack

    )

}

composable(Login) {

    LoginView(

        home = actions.home,

        back = actions.navigateBack

    )

}

composable(Home) {

    HomeView()

}

}

```

```
}
```

Creating Constants Object :

```
package com.project.pradyotprakash.flashchat

object Constants {

    const val TAG = "flash-chat"

    const val MESSAGES = "messages"

    const val MESSAGE = "message"

    const val SENT_BY = "sent_by"

    const val SENT_ON = "sent_on"

    const val IS_CURRENT_USER = "is_current_user"

}
```

Creating Navigation.Kt In Nav Package :

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

import androidx.navigation.NavHostController

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

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

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



```

/**
 * A set of destination used in the whole application
 */

object Destination {

    const val AuthenticationOption = "authenticationOption"

    const val Register = "register"

    const val Login = "login"

    const val Home = "home"

}

/**
 * Set of routes which will be passed to different composable so that
 * the routes which are required can be taken.
 */

class Action(navController: NavHostController) {

    val home: () -> Unit = {

        navController.navigate(Home) {

            popUpTo(Login) {

                inclusive = true

            }

            popUpTo(Register) {

```

```

        inclusive = true

    }

}

}

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

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

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

}

```

Creating View Package :

```

package com.project.pradyotprakash.flashchat.view

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.fillMaxHeight

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.*

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

```

```

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

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

/**
 * The authentication view which will give the user an option to choose between
 * login and register.
 */

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

    FlashChatTheme {

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

        Surface(color = MaterialTheme.colors.background) {

            Column(

                modifier = Modifier

                    .fillMaxWidth()

                    .fillMaxHeight(),

                horizontalAlignment = Alignment.CenterHorizontally,

                verticalArrangement = Arrangement.Bottom
            )
        }
    }
}

```

```

    ) {

        Title(title = " ⚡ Chat Connect")

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

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

    }

}

}

}

```

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

```
import androidx.compose.foundation.layout.fillMaxHeight
```

```
import androidx.compose.foundation.layout.fillMaxWidth
```

```
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.shape.RoundedCornerShape
```

```
import androidx.compose.foundation.text.KeyboardOptions
```

```
import androidx.compose.material.*
```

```
import androidx.compose.material.icons.Icons
```

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

```
import androidx.compose.runtime.Composable
```

```

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

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

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

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

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

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import com.project.pradyotprakash.flashchat.Constants

/**
 * Set of widgets/views which will be used throughout the application.
 * This is used to increase the code usability.
 */

@Composable
fun Title(title: String) {

    Text(

        text = title,

        fontSize = 30.sp,

        fontWeight = FontWeight.Bold,

```

```

        modifier = Modifier.fillMaxHeight(0.5f)

    )

}

// Different set of buttons in this page

@Composable

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

    Button(

        onClick = onClick,

        colors = ButtonDefaults.buttonColors(

            backgroundColor = backgroundColor,

            contentColor = Color.White

        ),

        modifier = Modifier.fillMaxWidth(),

        shape = RoundedCornerShape(0),

    ) {

        Text(

            text = title

        )

    }

}

```

```

@Composable

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

    TopAppBar(

        title = {

            Text(text = title)

        },

        navigationIcon = {

            IconButton(

                onClick = action

            ) {

                Icon(

                    imageVector = Icons.Filled.ArrowBack,

                    contentDescription = "Back button"

                )

            }

        }

    )

}

```

@Composable

```
fun TextFormField(value: String, onValueChange: (String) -> Unit, label: String,  
keyboardType: KeyboardType, visualTransformation: VisualTransformation) {
```

```
    OutlinedTextField(
```

```
        value = value,
```

```
        onValueChange = onValueChange,
```

```
        label = {
```

```
            Text(
```

```
                label
```

```
            )
```

```
        },
```

```
        maxLines = 1,
```

```
        modifier = Modifier
```

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

```
            .fillMaxWidth(),
```

```
        keyboardOptions = KeyboardOptions(
```

```
            keyboardType = keyboardType
```

```
        ),
```

```
        singleLine = true,
```

```
        visualTransformation = visualTransformation
```

```
    )
```



```

    }

    @Composable

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

        Card(

            shape = RoundedCornerShape(16.dp),

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

        ) {

            Text(

                text = message,

                textAlign =

                if (isCurrentUser)

                    TextAlign.End

                else

                    TextAlign.Start,

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

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

            )

        }

    }

```

Creating Home Package In View Package :

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

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

```
/**
```

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

```
 *
```

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

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

```
 */
```

```
@Composable
```

```
fun HomeView(
```

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

```
) {
```

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

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

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

```
)
```

```
Column(
```

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

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

```
    verticalArrangement = Arrangement.Bottom
```

```
) {
```

```
    LazyColumn(
```

```

modifier = Modifier

    .fillMaxWidth()

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

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

verticalArrangement = Arrangement.spacedBy(4.dp),

reverseLayout = true

) {

    items(messages) { message ->

        val isCurrentUser = message[Constants.IS_CURRENT_USER] as Boolean

        SingleMessage(

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

            isCurrentUser = isCurrentUser

        )

    }

}

OutlinedTextField(

    value = message,

    onValueChange = {

        homeViewModel.updateMessage(it)

    },

    label = {

        Text(

```

```

        "Type Your Message"

    )

},

maxLines = 1,

modifier = Modifier

    .padding(horizontal = 15.dp, vertical = 1.dp)

    .fillMaxWidth()

    .weight(weight = 0.09f, fill = true),

keyboardOptions = KeyboardOptions(

    keyboardType = KeyboardType.Text

),

singleLine = true,

trailingIcon = {

    IconButton(

        onClick = {

            homeViewModel.addMessage()

        }

    ) {

        Icon(

            imageVector = Icons.Default.Send,

            contentDescription = "Send Button"

        )

    }

}

```

```

        }
    )
}
}

```

```
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 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(),  
            verticalAlignment = Alignment.CenterVertically,  
            horizontalAlignment = Alignment.CenterHorizontally,  
            spacing = 100px  
        ) {  
            Text(email, style = TextStyle(fontSize = 16px))  
            Text(password, style = TextStyle(fontSize = 16px))  
            Text(loginText, style = TextStyle(fontSize = 16px))  
            Text(backText, style = TextStyle(fontSize = 16px))  
            Text(homeText, style = TextStyle(fontSize = 16px))  
        }  
    }  
}
```

```

modifier = Modifier.fillMaxSize(),

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Top
) {

AppBar(

    title = "Login",

    action = back

)

TextFormField(

    value = email,

    onChange = { loginViewModel.updateEmail(it) },

    label = "Email",

    keyboardType = TextInputType.Email,

    visualTransformation = VisualTransformation.None

)

TextFormField(

    value = password,

    onChange = { loginViewModel.updatePassword(it) },

    label = "Password",

    keyboardType = TextInputType.Password,

    visualTransformation = PasswordVisualTransformation()

)

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

```

```

Buttons(
    title = "Login",
    onClick = { loginViewModel.loginUser(home = home) },
    backgroundColor = Color.Magenta
)
}
}
}

```

```

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

```

```

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException

```

```

/**

```

```

 * 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 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,
        onChange = { registerViewModel.updateEmail(it) },
        label = "Email",
        keyboardType = TextInputType.Email,
        visualTransformation = VisualTransformation.None
    )

    TextFormField(
        value = password,
        onChange = { registerViewModel.updatePassword(it) },
        label = "Password",
        keyboardType = TextInputType.Password,
        visualTransformation = PasswordVisualTransformation()
    )

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

    Buttons(
        title = "Register",
        onClick = { registerViewModel.registerUser(home = home) },
        backgroundColor = Color.Blue
    )
}

```

```

    }
}

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

    }

}

}
```