

CHATCONNECT-A REALTIME CHAT AND COMMUNICATION APP

A PROJECT

Submitted by

P.ESAKKIRAJ (20201231506213)

S.ARUNTHAMILAN (20201231506204)

J.ASHOK BHARATHI (20201231506205)

K.KARTHICK (20201231506217)

MENTOR

Dr.T.ARUL RAJ M.Sc.,M.Phil.,Ph.D

in partial fulfillment of the requirements for the award of degree of

BACHELOR OF SCIENCE



SRI PARAMAKALYANI COLLEGE

ALWARKURICHI – 627 412

APRIL - 2023

TABLE OF CONTENTS

Chapter No.	Title	Page No.
I	INTRODUCTION	
	1.1 Overview 1.2 Purpose	
II	PROBLEM DEFINITION & DESIGN THINKING	
	2.1 Empathy Map 2.2 Ideation & Brainstorming Map	
III	RESULT	8
IV	ADANTAGES & DISADANTAGES	11
V	APPLICATIONS	12
VI	CONCLUSION	13
VII	FUTURE SCOPE	13
	APPENDIX	
	(i) Source Code	

CHAPTER –I 1.INTRODUCTION:

1.1 OVERVIEW

Our project is related to a new way of chatting with people. Chatting and communicating with people through internet is becoming common to people and is connecting people all over the world.

Mainly, chatting apps in today's world mainly focus on connecting people, providing users with more features like GIFs, stickers etc. But this app, is different from them.

This chatting application includes chatting through internet using IPaddress. It mainly focuses on chatting and connects people all around the world.

Mostly, chatting applications like WhatsApp requires mobile no. of the person and then we can chat and connect with the person. But here, the person only has to login with the system, and then he can connect with the people which he wants with.

ChatConect – A Real-Time Chat and Communication App -It means people all over the world can join the chat between people easily. We can check and see the people joining and leaving the chat group.

For using the app, firstly we have to register our name in the application. After registration, the person will be given a particular IP address, which is only used by that person, so that people with same name can be differentiated easily.

The IP address can only be seen by the person which is registered under that name. Once, the registration of the person is done, he can join the

chat room. Chatapp is a digital communication platform that allows users to exchange messages and have real-time conversations.

It is a software application designed to provide instant messaging services on various devices, including smartphones, tablets, and computers.

Chatapp has become a popular way to communicate with friends, family, and colleagues because of its convenience, speed, and accessibility.

Chatapp typically allows users to create profiles, add friends or contacts, and join or create groups or chat rooms. Users can send text messages, emojis, images, and videos to individuals or groups.

Many chatapps also offer additional features, such as voice and video calls, file sharing, and integration with other applications.

Chatapp has become an essential tool for personal and business communication, as it allows people to stay connected regardless of their location or time zone. It has also become a popular platform for social networking, dating, and online communities.

Chatting is a method of using technology to bring people and ideas “together” despite of the geographical barriers.

The technology has been available for years but the acceptance it was quite recent. Our project is an example of a chat server.

The aim of this project is to build a functional real-time messaging application for developers by using modern web technologies.

1.2 PURPOSE

The purpose of this project is to design a chat application, also known as a instant messaging system.

The main purpose of the software is to provide users with an instant messaging tool that has the ability to handle millions of users simultaneously when needed and can be easily done.

The purpose of chat communication is to facilitate real- time conversations between individuals or groups, whether for personal or professional reasons.

Many chat communication apps also support the sharing of multimedia content like images and videos, making it easier to convey ideas and emotions.

Chat communication allows people to connect with one another quickly and easily, regardless of geographic location, making it an essential tool for remote work and long-distance relationships.

Chat communication is also a more informal and relaxed way of communicating compared to email or phone calls, making it ideal for casual conversations, quick updates, and brainstorming sessions.

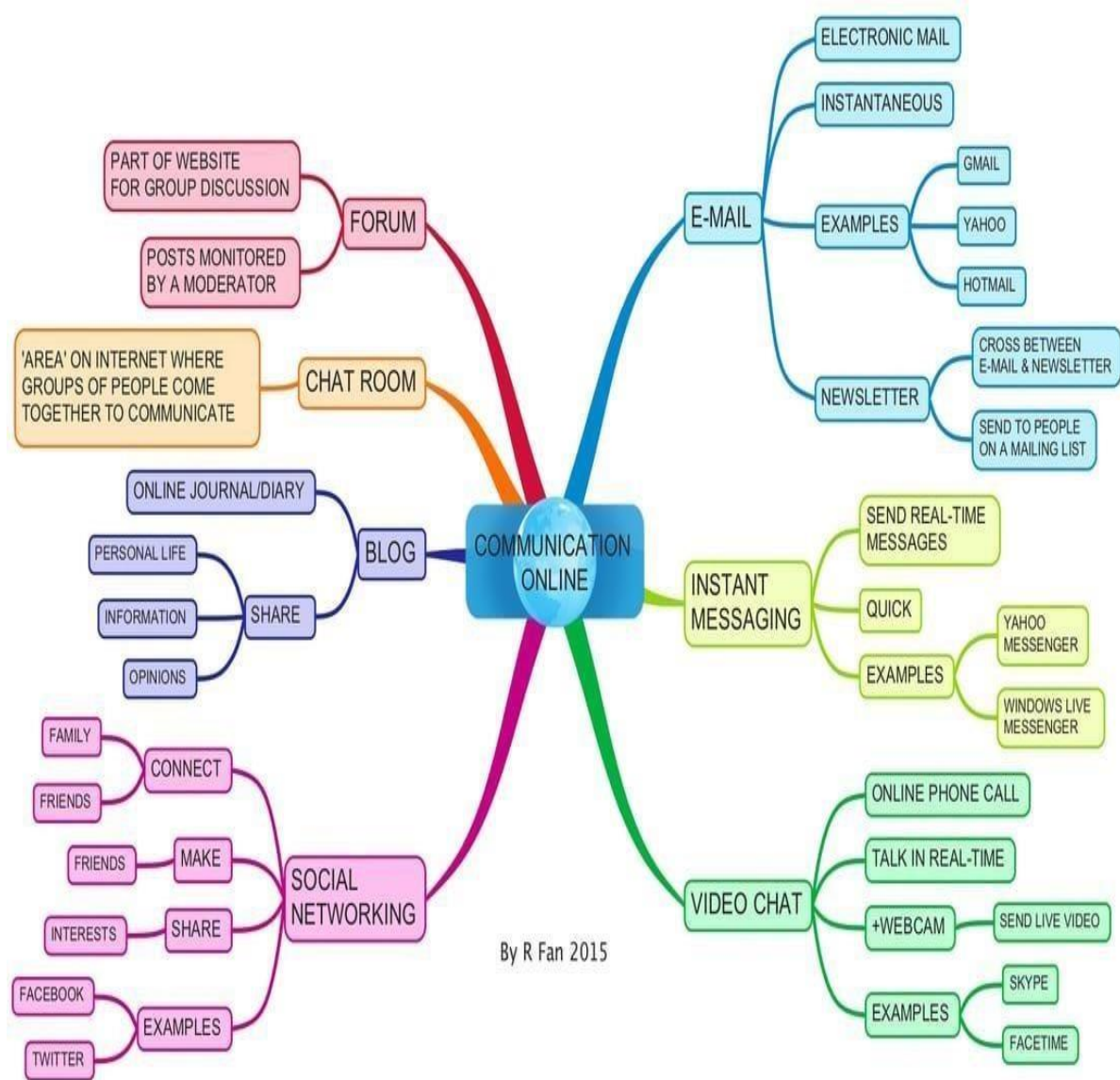
CHAPTER–II PROBLEM DEFINITION & THINKING

2.1 EMPATHY MAP

This is sample text, enter your text here.



2.2 IDEATION & BRAINSTORMING MAP



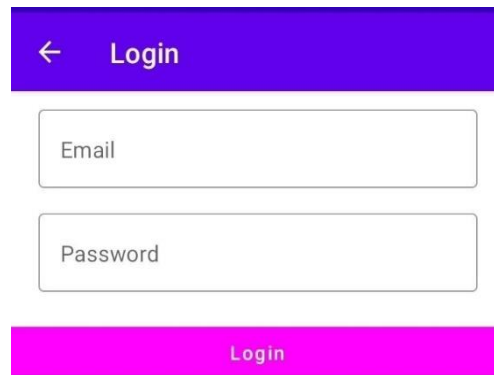
CHAPTER – III 3.1 RESULT:
REGISTER PAGE

 Chat Connect

Register

Login

LOGIN PAGE :



The image shows a mobile app login screen. At the top is a blue header bar with a white back arrow and the text 'Login'. Below the header are two white input fields with rounded corners. The first field is labeled 'Email' and the second is labeled 'Password'. At the bottom of the form is a red button with the text 'Login' in white. The entire form is centered on a white background.

← Login


Email

Password

Login

MESSAGE PAGE:



Type Your Message 

CHAPTER – IV ADVANTAGES & DISADVANTAGES:

ADVANTAGES:

Instant communication: Chat communication apps allow you to instantly communicate with others, whether it's one-on-one or in a group setting.

Efficiency: With the ability to share multimedia content and real-time collaboration features, chat communication apps can increase efficiency in communication and project management.

Cost-effective: Many chat communication apps are free or low-cost, making them a cost-effective alternative to traditional communication methods.

Flexibility: Chat communication apps can be used on various devices, including smartphones, tablets, and computers, making them a flexible and accessible communication option.

Privacy and security: Many chat communication apps offer end-to-end encryption and other security features to protect user privacy and Sensitive information.

Disadvantages:

Miscommunication: With the lack of nonverbal cues and tone of voice, miscommunication can occur in chat communication, leading to misunderstandings and conflicts.

Distractions: Notifications and constant messages can be distracting and disruptive to productivity and focus.

Limited context: Chat communication can be limited in providing context and detail, leading to incomplete or inaccurate information sharing.

Dependence on technology: Dependence on chat communication apps can lead to a lack of face-to-face interaction and social skills development.

Overuse: Overuse of chat communication apps can lead to burnout and negative impacts on mental health and well-being.

CHAPTER – V APPLICATION:

Personal Applications:

Keeping in touch with friends and family who live far away.

Coordinating plans and events with a group of friends or family members.

Sharing updates, photos, and videos with loved ones.

Meeting new people with similar interests and engaging in online communities.

Professional Applications:

Internal team communication for remote or distributed teams.

Collaboration on projects and tasks with colleagues and team members.

Communication with clients and customers for customer support and sales.

Coordination of meetings and scheduling with team members and clients.

Sharing of files, documents, and other important information among team members.

Facilitating communication between different departments and stakeholders in an organization.

Communication with vendors and suppliers for procurement and supply chain management.

Overall, chat communication apps can be applied in a variety of contexts to improve communication, collaboration, and efficiency.

CHAPTER – VI CONCLUSION

In conclusion, chat communication apps offer a convenient and efficient way to communicate with others in both personal and professional settings. With features like real-time messaging, multimedia sharing, and collaboration tools, chat communication apps can enhance productivity and streamline communication. While there are some potential drawbacks to using these apps, such as the risk of miscommunication and distractions, the benefits often outweigh the negatives. With the continued growth and development of these apps, we can expect them to become an increasingly important part of our communication and collaboration workflows.

CHAPTER – VII FUTURE SCOPE

CHAT COMMUNICATION

The future scope of chat communication apps is quite vast and exciting, as technology continues to advance and evolve. Here are a few potential directions in which chat communication apps could develop in the future

Integration with AI: Chat communication apps could integrate with artificial intelligence (AI) to offer more personalized and intelligent responses. This could include features like chatbots, voice assistants, and machine learning algorithms that can understand user preferences and offer tailored recommendations.

Augmented reality: With the development of augmented reality (AR) technology, chat communication apps could incorporate AR features that allow users to interact with each other in new and exciting ways. For example, users could use AR to play games together or to try on virtual clothing before making a purchase.

Privacy and security: With concerns around privacy and security increasing, chat communication apps could prioritize security features like end-to-end encryption, two-factor authentication, and data protection measures. Users are increasingly concerned about their personal data being

shared or stolen, and chat communication apps that prioritize privacy and security could have a competitive advantage in the future

CHAPTER – VIII APPENDIX

A.SOURCE CODE

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.

15

*/

```
Class MainActivity : ComponentActivity() {
```

```
    Override fun onCreate(savedInstanceState: Bundle?) {
```

```
        Super.onCreate(savedInstanceState)
```

```
        FirebaseApp.initializeApp(this)
```

```
        setContent {
```

```
            NavComposeApp()
```

```
        }
```

```
    }
```

```
}
```

Nav Comose App.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()

}
```

```

    }

}

}

```

Constant Object.kt File

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 file

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() }

}

```

VIEW PACKAGE

AuthenticationOption.kt File

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)

    }

}

}

}

```

Widgets.kt File

```

Package com.project.pradyotprakash.flashchat.view

Import androidx.compose.foundation.layout.fillMaxHeight

Import androidx.compose.foundation.layout.fillMaxWidth

Import androidx.compose.foundation.layout.padding

```

Import androidx.compose.foundation.shape.RoundedCornerShape

Import androidx.compose.foundation.text.KeyboardOptions

Import androidx.compose.material.*

Import androidx.compose.material.icons.Icons

Import androidx.compose.material.icons.filled.ArrowBack

Import androidx.compose.runtime.Composable

Import androidx.compose.ui.Modifier

Import androidx.compose.ui.graphics.Color

Import androidx.compose.ui.text.font.FontWeight

Import androidx.compose.ui.text.input.KeyboardType

Import androidx.compose.ui.text.input.VisualTransformation

Import androidx.compose.ui.text.style.TextAlign

Import androidx.compose.ui.unit.dp

Import androidx.compose.ui.unit.sp

Import com.project.pradyotprakash.flashchat.Constants

/**

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

This is used to increase the code usability.


```
*/
```

```
@Composable
```

```
Fun Title(title: String) {
```

```
    Text(
```

```
        Text = title,
```

```
        fontSize = 30.sp,
```

```
        fontWeight = FontWeight.Bold,
```

```
        modifier = Modifier.fillMaxHeight(0.5f)
```

```
    )
```

```
}
```

```
// Different set of buttons in this page
```

```
@Composable
```

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

```
    Button(
```

```
        onClick = onClick,
```

```
        colors = ButtonDefaults.buttonColors(
```

```
            backgroundColor = backgroundColor,
```

```
            contentColor = Color.White
```

```

    ),

    Modifier = Modifier.fillMaxWidth(),

    Shape = RoundedCornerShape(0),

) {

    Text(

        Text = title

    )

}

}

@Composable

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

    TopAppBar(

        Title = {

            Text(text = title)

        },

        navigationIcon = {

            IconButton(

                onClick = action

            )

```

```

    ) {

        Icon(

            imageVector = Icons.Filled.ArrowBack,

            contentDescription = "Back button"

        )

    }

}

)

}

@Composable

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

    OutlinedTextField(

        Value = value,

        onValueChange = onValueChange,

        label = {

            Text(

                Label

            )

        }

```

```

    },

    maxLines = 1,

    modifier = Modifier

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

        .fillMaxWidth(),

    keyboardOptions = KeyboardOptions(

        keyboardType = keyboardType

    ),

    singleLine = true,

    visualTransformation = visualTransformation

)

}

@Composable

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

    Card(

        Shape = RoundedCornerShape(16.dp),

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

    ) {

```

```

Text(

    Text = message,

    textAlign =

    if (isCurrentUser)

        TextAlign.End

    Else

        TextAlign.Start,

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

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

)

}

}

```

Home.kt File

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

```
Import androidx.compose.foundation.background
```

```
Import androidx.compose.foundation.layout.*
```

```
Import androidx.compose.foundation.lazy.LazyColumn
```

```
Import androidx.compose.foundation.lazy.items
```

Import androidx.compose.foundation.text.KeyboardOptions

Import androidx.compose.material.*

Import androidx.compose.material.icons.Icons

Import androidx.compose.material.icons.filled.Send

Import androidx.compose.runtime.Composable

Import androidx.compose.runtime.getValue

Import androidx.compose.runtime.livedata.observeAsState

Import androidx.compose.ui.Alignment

Import androidx.compose.ui.Modifier

Import androidx.compose.ui.graphics.Color

Import androidx.compose.ui.text.input.KeyboardType

Import androidx.compose.ui.unit.dp

Import androidx.lifecycle.viewmodel.compose.viewModel

Import com.project.pradyotprakash.flashchat.Constants

Import com.project.pradyotprakash.flashchat.view.SingleMessage

/**

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

*

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

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

```
*/
```

```
@Composable
```

```
Fun HomeView(
```

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

```
) {
```

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

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

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

```
)
```

```
Column(
```

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

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

```
    verticalArrangement = Arrangement.Bottom
```

```
) {
```

```
    LazyColumn(
```

```
        Modifier = Modifier
```

```

        .fillMaxWidth()

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

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

        verticalArrangement = Arrangement.spacedBy(4.dp),

        reverseLayout = true

    ) {

        Items(messages) { message ->

            Val isCurrentUser = message[Constants.IS_CURRENT_USER] as Boolean

            SingleMessage(

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

                isCurrentUser = isCurrentUser

            )

        }

    }

    OutlinedTextField(

        Value = message,

        onChange = {

            homeViewModel.updateMessage(it)

```



```

    },

    Label = {

        Text(

            "Type Your Message"

        )

    },

    maxLines = 1,

    modifier = Modifier

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

        .fillMaxWidth()

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

    keyboardOptions = KeyboardOptions(

        keyboardType = KeyboardType.Text

    ),

    singleLine = true,

    trailingIcon = {

        IconButton(

            onClick = {

```

```
        homeViewModel.sendMessage()

    }

) {

    Icon(

        imageVector = Icons.Default.Send,

        contentDescription = "Send Button"

    )

}

}

)

}

}
```

Homeview Model.kt File

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()

}

}

```

Login.kt File

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 = TextInputType.Email,

        visualTransformation = VisualTransformation.None

    )

    TextFormField(

        Value = password,
```

```
        onChange = { loginViewModel.updatePassword(it) },

        label = "Password",

        keyboardType = KeyboardType.Password,

        visualTransformation = PasswordVisualTransformation()

    )

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

    Buttons(

        Title = "Login",

        onClick = { loginViewModel.loginUser(home = home) },

        backgroundColor = Color.Magenta

    )

}

}
```

LoginviewModel.kt File

Package com.project.pradyotprakash.flashchat.view.login

Import androidx.lifecycle.LiveData

Import androidx.lifecycle.MutableLiveData

Import androidx.lifecycle.ViewModel

Import com.google.firebase.auth.FirebaseAuth

Import com.google.firebase.auth.ktx.auth

Import com.google.firebase.ktx.Firebase

Import java.lang.IllegalArgumentException

/**

- View model for the login view.

*/

```
Class LoginViewModel : ViewModel() {

    Private val auth: FirebaseAuth = Firebase.auth

    Private val _email = MutableLiveData("")

    Val email: LiveData<String> = _email

    Private val _password = MutableLiveData("")

    Val password: LiveData<String> = _password

    Private val _loading = MutableLiveData(false)

    Val loading: LiveData<Boolean> = _loading

    // Update email

    Fun updateEmail(newEmail: String) {

        _email.value = newEmail
```

```
}
```

```
// Update password
```

```
Fun updatePassword(newPassword: String) {
```

```
    _password.value = newPassword
```

```
}
```

```
// Register user
```

```
Fun loginUser(home: () -> Unit) {
```

```
    If (_loading.value == false) {
```

```
        Val email: String = _email.value ?: throw IllegalArgumentException("email expected")
```

```
        Val password: String =
```

```
            _password.value ?: throw IllegalArgumentException("password expected")
```

```
        _loading.value = true
```



```
Auth.signInWithEmailAndPassword(email, password)

    .addOnCompleteListener {

        If (it.isSuccessful) {

            Home()

        }

        _loading.value = false

    }

}

}

}
```

Register.kt File

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

```
Import androidx.compose.foundation.layout.*
```

```
Import androidx.compose.material.CircularProgressIndicator
```

```
Import androidx.compose.runtime.Composable
```

```
Import androidx.compose.runtime.getValue
```

```
Import androidx.compose.runtime.livedata.observeAsState
```

```
Import androidx.compose.ui.Alignment
```

```
Import androidx.compose.ui.Modifier
```

```
Import androidx.compose.ui.graphics.Color
```

```
Import androidx.compose.ui.text.input.KeyboardType
```

```
Import androidx.compose.ui.text.input.PasswordVisualTransformation
```

```
Import androidx.compose.ui.text.input.VisualTransformation
```

Import androidx.compose.ui.unit.dp

Import androidx.lifecycle.viewmodel.compose.viewModel

Import com.project.pradyotprakash.flashchat.view.Appbar

Import com.project.pradyotprakash.flashchat.view.Buttons

Import com.project.pradyotprakash.flashchat.view.TextFormField

/**

- The Register view which will be helpful for the user to register themselves into
- Our database and go to the home screen to see and send messages.

*/

@Composable

Fun RegisterView(

Home: () -> Unit,

Back: () -> Unit,

```

registerViewModel: RegisterViewModel = viewModel()

) {

    Val email: String by registerViewModel.email.observeAsState("")

    Val password: String by registerViewModel.password.observeAsState("")

    Val loading: Boolean by registerViewModel.loading.observeAsState(initial = false)

Box(

    contentAlignment = Alignment.Center,

    modifier = Modifier.fillMaxSize()

) {

    If (loading) {

        CircularProgressIndicator()

    }

    Column(

        Modifier = Modifier.fillMaxSize(),

```

```

        horizontalAlignment = Alignment.CenterHorizontally,

        verticalArrangement = Arrangement.Top

    ) {

        AppBar(

            Title = "Register",

            Action = back

        )

        TextFormField(

            Value = email,

            onValueChange = { registerViewModel.updateEmail(it) },

            label = "Email",

            keyboardType = TextInputType.Email,

            visualTransformation = VisualTransformation.None

        )

        TextFormField(

```

```
        Value = password,

        onValueChange = { registerViewModel.updatePassword(it) },

        label = "Password",

        keyboardType = KeyboardType.Password,

        visualTransformation = PasswordVisualTransformation()

    )

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

    Buttons(

        Title = "Register",

        onClick = { registerViewModel.registerUser(home = home) },

        backgroundColor = Color.Blue

    )

}

}

}
```

Registerview Model.kt File

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
```

```
        }
```

```
    }
```

```

    }

}

```

Modifying AndroidManifest.xml File

```

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    Package="com.project.pradyotprakash.flashchat">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application

        Android:allowBackup="true"
        Android:icon="@mipmap/ic_launcher"
        Android:label="@string/app_name"
        Android:roundIcon="@mipmap/ic_launcher_round"
        Android:supportsRtl="true"
        Android:theme="@style/Theme.FlashChat">

        <activity

            Android:name=".MainActivity"
            Android:exported="true"
            Android:label="@string/app_name"
            Android:theme="@style/Theme.FlashChat.NoActionBar">
            <intent-filter>

```

```
<action android:name="android.intent.action.MAIN" />
```

```
<category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>
```

```
</activity>
```

```
</application>
```

```
</manifest>
```