CHAT CONNECT: A REAL-TIME CHAT AND COMMUNICATION APP

**V.JAHANKIR KABEER**

**K.YOGESH**

**S.MOHAMED HISHAM**

**B.ARUN**

**M.MOHAMED ALHAITH**

## 1. INTRODUCTION

## *1.1 Overview*

ChatConnect is a sample project built using the Android Compose UI toolkit. It demonstrates how to create a simple chat app using the Compose libraries. The app allows users to send and receive text messages. The project 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.

**Project Workflow:**

* Users register into the application.

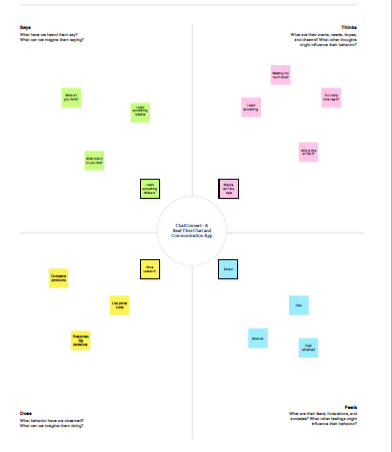
* After registration, user logins into the application.
* User enters into the main page

*1.2 Purpose*

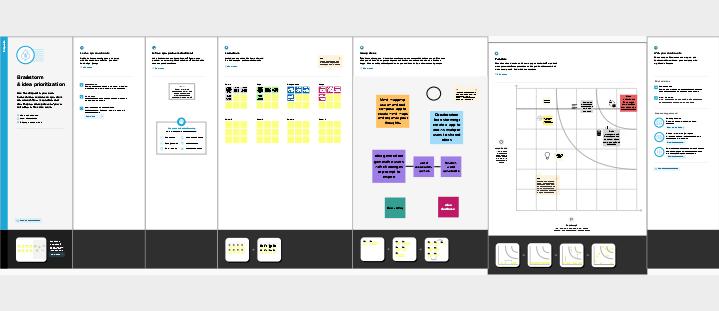
The purpose of a demonstration of Chatting app allows you to communicate with your customers Web Chat Rooms. It enables you to send and receive messages. Chatting apps make it easier, simpler, and faster to connect with everyone and it is also easy to use. There are many types of chatting apps and every one has its own format, design, and functions. A great example here is the Chat Connect messaging app.

**2. Purpose Definition & Design Thinking**

*2.1 Empathy map*

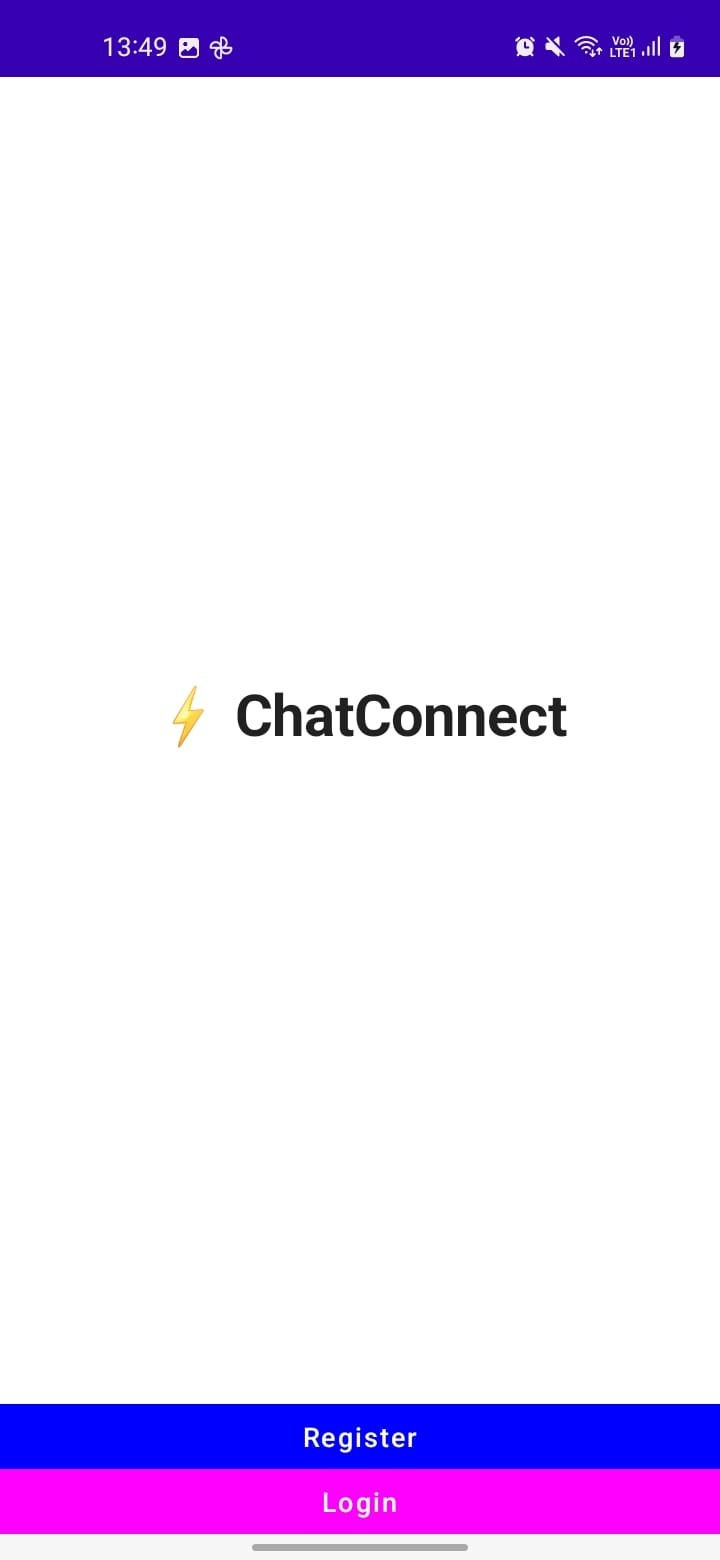


*2.2 Ideation & Brainstorming Map*

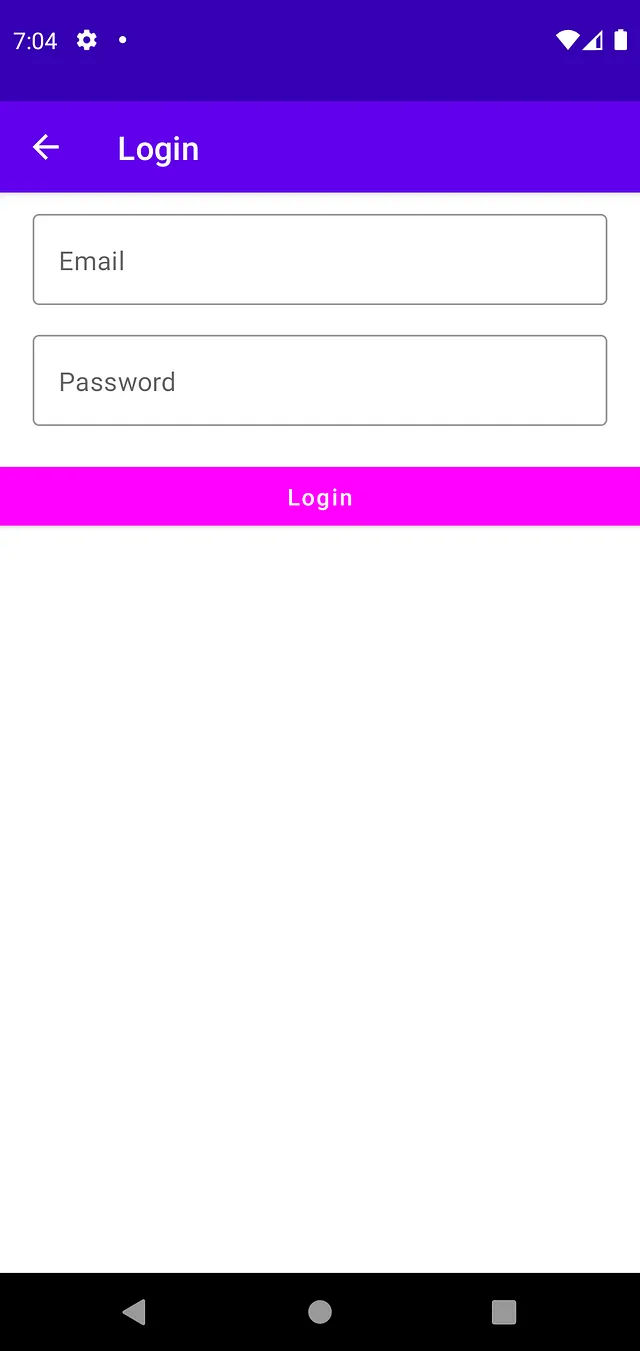


**3. RESULT**

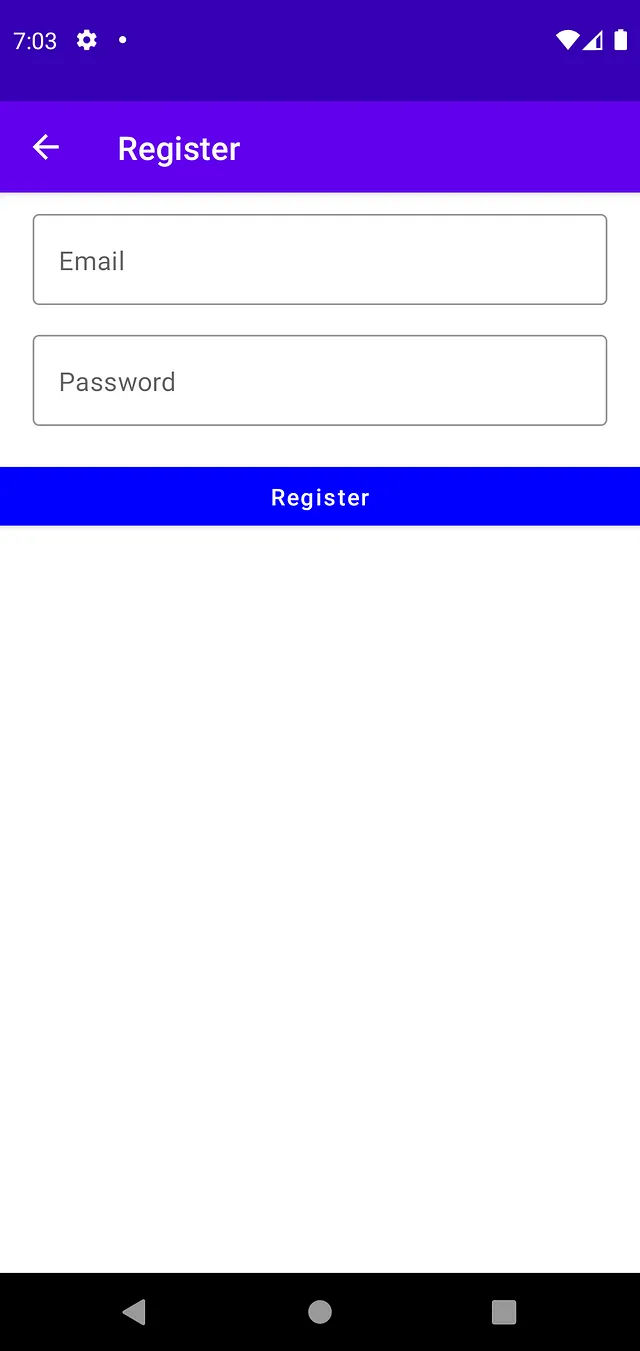
First page:



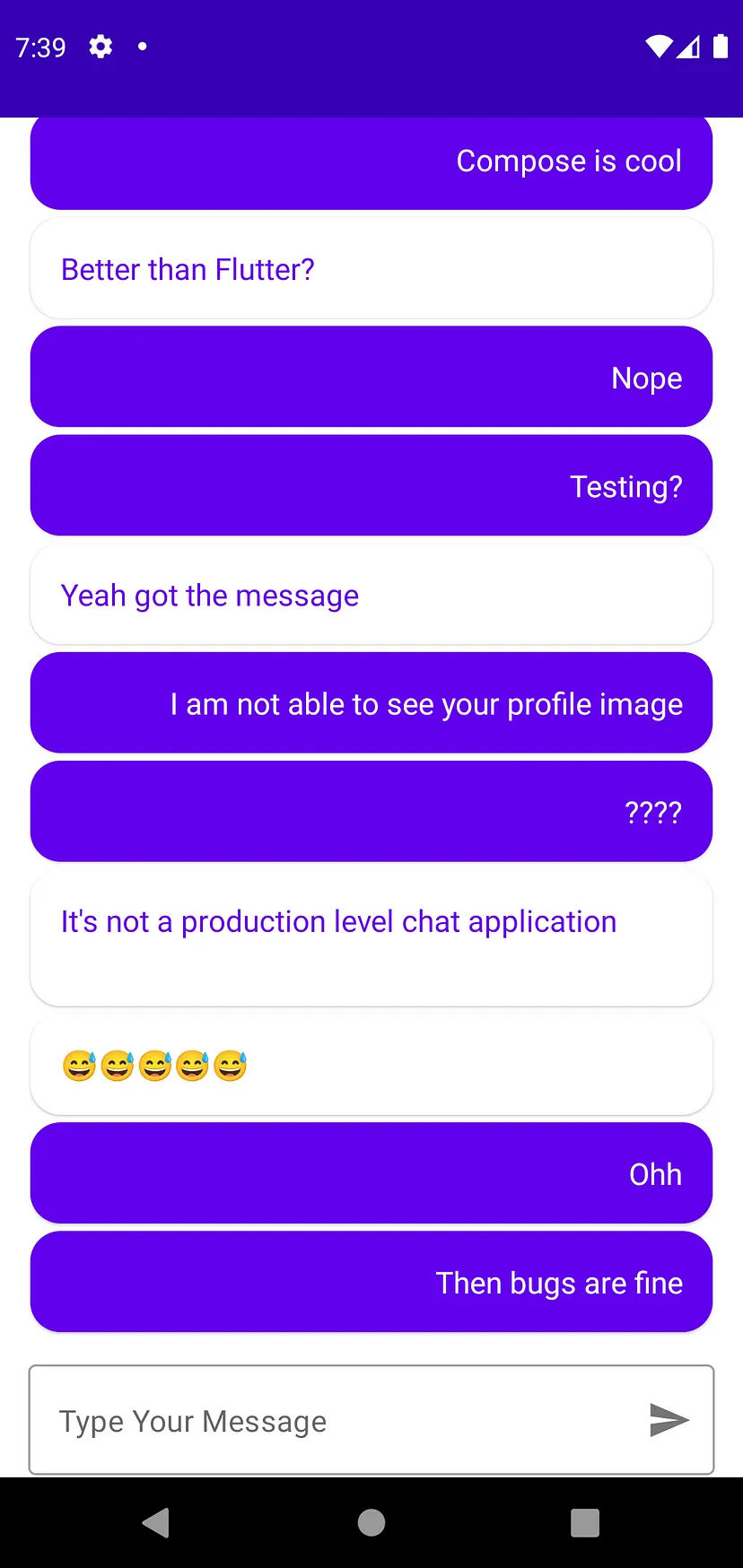
Login Page:



Register Page:



Main Page:



**4 ADVANTAGES AND DISADVANTAGES**

*Advantages:*

Simplified syntax: Compose offers a simplified syntax for defining UI elements, including text inputs, that makes it easier to create and maintain code.

Real-time preview: Compose offers a real-time preview of your UI as you write your code, allowing you to see changes as you make them.

Automatic validation: Compose provides built-in support for input validation, making it easy to ensure that user input meets your application's requirements.

Improved performance: Compose uses a more efficient rendering system, resulting in faster UI rendering and improved performance.

*Disadvantages:*

Learning curve: Compose is a new technology, so there is a learning curve for developers who are new to it.

Limited documentation: Compose is still in its early stages, so there may be limited documentation and resources available.

Compatibility: Compose is only supported on newer versions of Android, so older devices may not be able to run applications built with Compose.

Complexity: While Compose simplifies many aspects of UI development, more complex UI elements can still be challenging to create.

**5 APPLICATIONS**

First screens: The First Screen Asking For Register or Login.

Login screens: User authentication is a common feature in many Android apps, and login screens typically require users to enter their email or username and password. Using the Text Input Field component with validation can help ensure that users enter valid login credentials before attempting to log in.

Registration screens: Registration screens often require users to enter their name, email, password, and other personal information. Using the Text Input Field component with validation can help ensure that the information entered is accurate and complete.

**6 Conclusion**

In conclusion, Android Compose provides a powerful and flexible way to handle text input and validation in your app. With Compose, you can easily create custom input fields and apply validation logic to ensure that user input is correct and consistent. You can also take advantage of Compose's state management and event handling features to update your UI in real-time as the user interacts with your app.

The chat app provides a better and more flexible chat system. Developed with the latest technology in the way of providing a reliable system. The main advantage of the system is instant messaging, real-world communication, added security, group chat, etc.

**7 FUTURE SCOPES**

1. A place of safety
2. Video call
3. large size
4. Conference call
5. Voice recording will be added
6. Improving different text style and font size.

**8 APPENDIX**

1. Source code

Step 1:

**Complete code for Mainactivity.kt**

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()

}

}

}

Step 2:

**Creating NavCompose.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()

}

}

}

}

Step 3:

**Create an 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"

}

Step 4:

Create Nav Package

**Create 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() }

}

Step 5:

Create View Package

**Create 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)

}

}

}

}

**Create 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

)

}

}

Step 6:

Create Home package in View package

**Create 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,

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"

)

}

}

)

}

}

**Create an HomeViewModel class**

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()

}

}

Step 7:

Create Login package in View package

**Create 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 = 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

)

}

}

}

**Create an LoginViewModel class**

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

}

}

}

}

Step 8:

Create Register package in View package

**Create 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 = 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

)

}

}

}

**Create an RegisterViewModel class**

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

}

}

}

}