**Chatconnect – A Real-Time chat and communication App**

**M. Sowmiya**

**K. Kavitha**

**K. Shruthisamyugtha**

**P. Vithya**

Introduction:

* 1. Overview

Chat Connect provides an interactive platform for users to engage in discussions, ask questions, or seek advice on various topics including but not limited to technology, education, health, entertainment, and lifestyle.

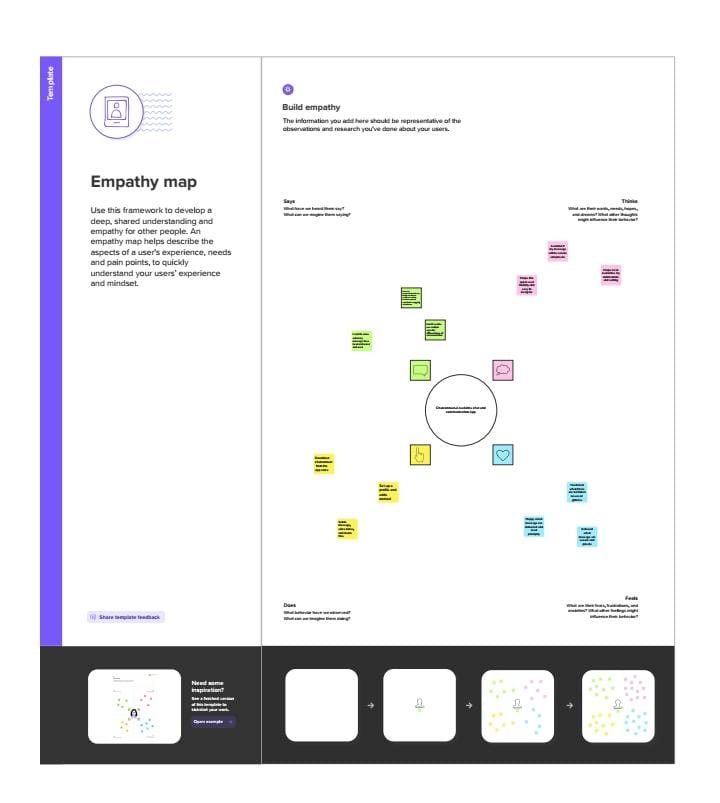
Project workflow

1. Project workflow refers to the sequence of steps and activities that are involved in completing a project. The workflow can vary depending on the nature od the project, the organization, and team involved.
2. Intiation: In this stage, the project idea is developed and refined, and the scope and objectives of the project are defined. This stage also involves determining the resources and team required to carry out the project
3. Planning: in this stage, a detailed project plan is developed that includes tasks, timelines, milestones, and resources allocation. This stage also involves identifying potential paln and developing contingency plans.
4. Execution: This Stage involves implementing the project plan, and the Task identified in the planning Stage are Carried out. Project manager oversee The work of the team , progress is tracked regularly to ensure that the Project is on track.

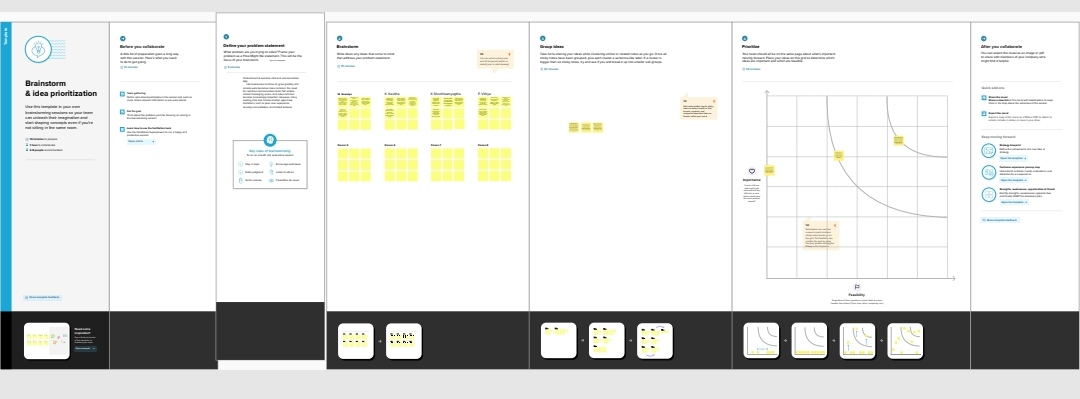
Overall, a well-defined Project workflow can help ensure that the project is completed on time, Within budget, and meat the requirement of the stakeholders.

* 1. **Purpose:**
* As an AI Language model, My purpose is To provide conversation and assistant to user Who interact with me. chat connect is not term that I’m Familiar with, So I am not sure if you are referring to a specific Context or platform.

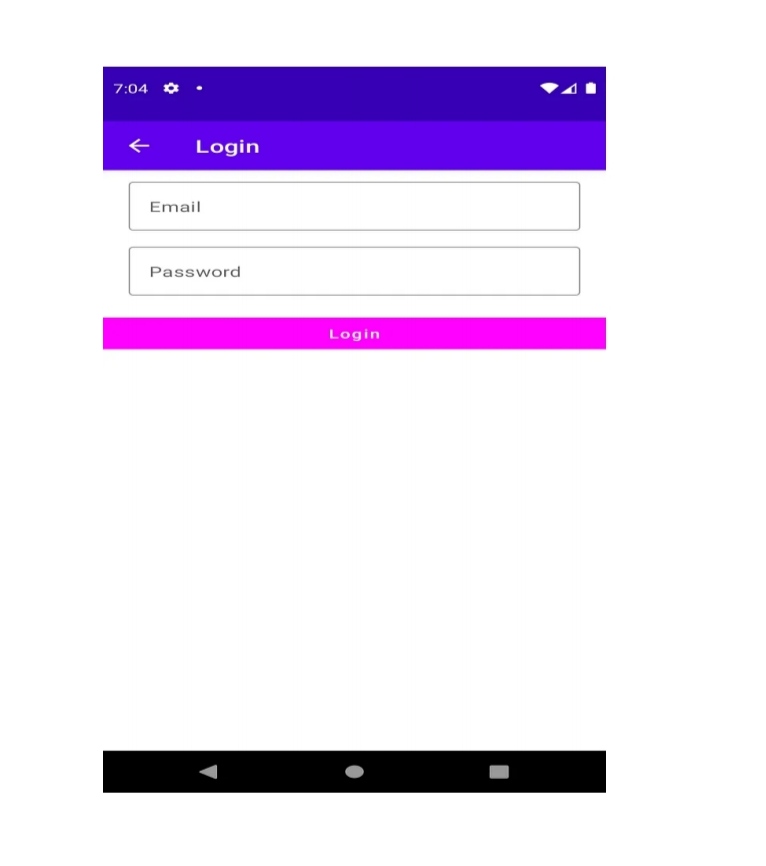
1. . **Problem definition and design thinking:**
   1. **Empathy map**



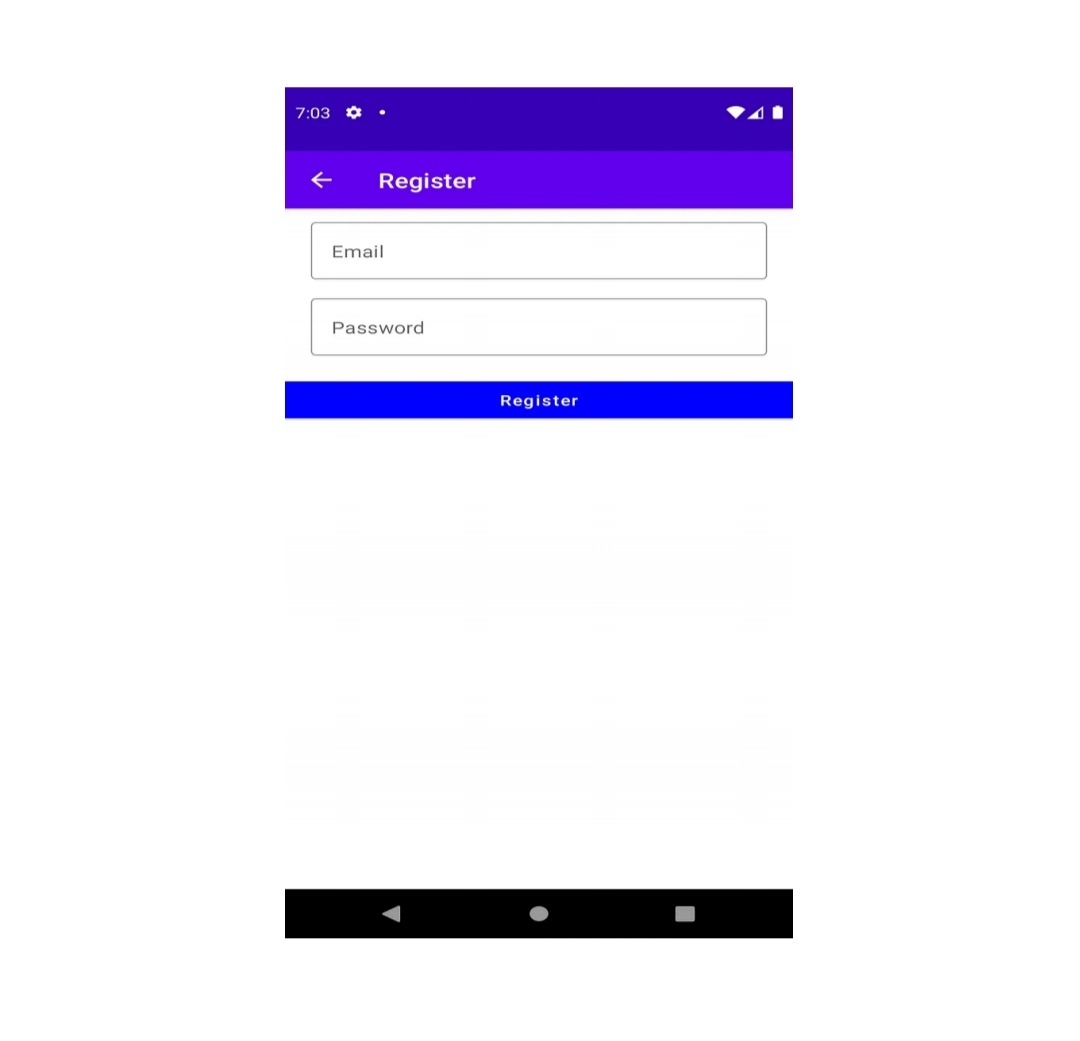
* 1. **Ideation and brainstorming:**

****

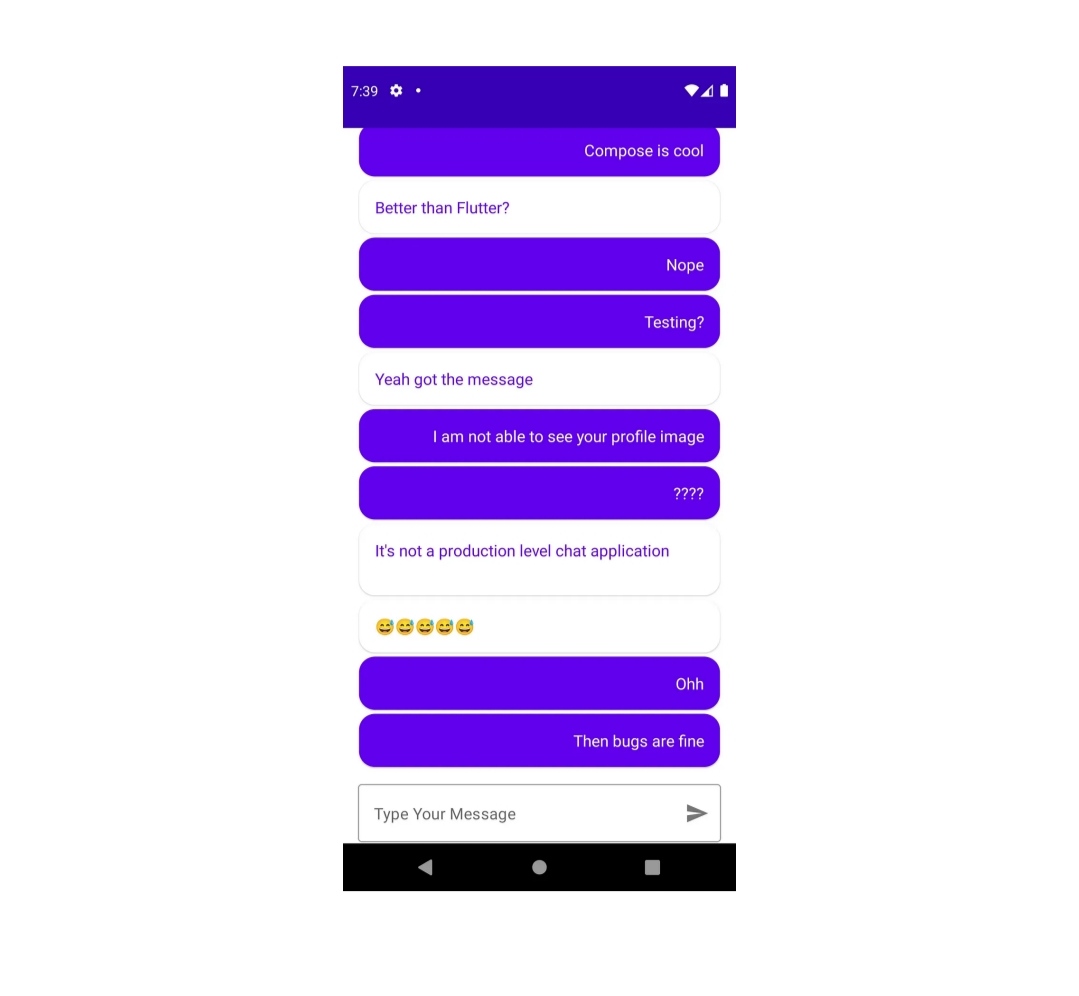
3.Result:

Login page:

Register page:



Home:



4.**Advantage and Disadvantage:**

**Advantages:**

Chat connect, I am not aware of any specific technology or service that goes by the name.

1. Convenience: Chat and messaging services are typically available 24/7, Making it easy to communicate with other at anytime and from any location
2. Speed: Chat and messaging services allow for fast communication, allowing user to quickly exchange information, ask questions,are share update.
3. Cost effective:Many chat and messaging services are free to use, making them and affordable option for business and individual
4. Privacy: Chat and messaging services typically offer end to end encryption and other security features to ensure that messages are kept private and confidential.

**Disadvantage:**

1. Security concerns: Many messaging apps may not have end-to-end encryption, which means that messages may be vulnerable to hacking or interception.
2. Privacy concerns: Some messaging apps may collect and use user data in ways that users may not be comfortable with.
3. Limited functionality: Some messaging apps may not have all the features that users want or need.
4. Reliance on internet connectivity: Messaging apps require a stable internet connection, which may not be available in all areas or at all times.
5. User adoption: If not many people are using the same messaging app as you, it may limit your ability to communicate with them.

**Application:**

* Chat applications allow you to stay connected with other people who may be using the application even on the other side of the world. In customer service, such applications are one of the most important communication channels.
* Chat applications typically run on centralized networks that are served by platform operator servers as opposed to peer-to-peer protocols such as XMPP. This allows two people to talk to each other in real time.
* Of course. We have many free chat apps available in the market. They are used both in everyday use and, for example, in customer service. An example of such an application is Messenger, the operation of which can be synchronized with the LiveAgent software**.**

**Conclusion:**

* + An app, short for “application,” is a software program designed to perform a specific function or set of functions. Apps can be used on various devices such as smartphones, tablets, and computers.
    - The way an app works depends on its purpose and design. Some apps are standalone programs, while others are designed to work in conjunction with other apps or services.
    - To define an app, you can describe it as a software program that performs a specific task or set of tasks. Apps can be designed for various purposes, such as entertainment, productivity, social networking, and communication. They can be downloaded and installed on a device, or accessed through a web browser.

**Future scope:**

* Integration with Virtual and Augmented Reality: As virtual and augmented reality technologies continue to advance, it may become possible to integrate Chat Connect into these platforms. This could allow users to interact with chatbots and other virtual assistants in a more immersive way.
* Personalized Conversational Interfaces: With the help of advanced machine learning and natural language processing techniques, Chat Connect could be personalized to better suit individual users’ needs and preferences. This could lead to more engaging and effective conversations.
* Integration with IoT Devices: As the Internet of Things (IoT) continues to grow, Chat Connect could be integrated with a wide range of IoT devices, including smart home appliances, wearable devices, and more. This could enable users to control their devices using natural language commands.
* Integration with Social Media Platforms: Chat Connect could be integrated with popular social media platforms like Facebook, Twitter, and Instagram, allowing users to chat with bots and other automated systems directly within these platforms.
* Advanced Security Features: As cyber threats become increasingly sophisticated, Chat Connect could be equipped with advanced security features like end-to-end encryption and biometric authentication to ensure users’ privacy and security.

Overall, the future scope for Chat Connect is vast and exciting, and there are many ways in which this technology can be leveraged to improve the way we communicate and interact with machines.

**8.Appendix**

**A. Source code**

Step no. 1 : NavcomposeApp

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 no. 2: Constant

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 3 : Navigation**

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 4 : Authentication**

**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)**

**}**

**}**

**}**

**}**

**Step 5 widgets**

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 Home**

**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”

)

}

}

)

}

}

Step 7 Home view model

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 8 Login

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

)

}

}

}

**Step 9 loginviewmodel**

**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 10 Register**

**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**

**)**

**}**

**}**

**}**

**Step 11 Registerviewmodel**

**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**

**}**

**}**

**}**

**}**