

# PROJECT REPORT

## ChatConnect- A Real-Time Chat and Communication App

**Team Leader** :Velayutham.K

**Team Members** :Shanmugam.P

Altrin.R

Gobi Silvan.Y

### 1.INTRODUCTION:

ChatConnect is a real-time chat and communication app designed to connect people around the world. It allows users to chat with one another in real-time, share files, make voice and video calls, and stay connected with their friends, family, and colleagues. The app is available on both desktop and mobile devices, making it easy to stay in touch on-the-go.

With ChatConnect, you can create private or group chats, share photos and videos, and send messages with emojis and stickers to express yourself. You can also make voice and video calls with other users, even if they are located in different parts of the world. ChatConnect also offers end-to-end encryption to ensure that your conversations remain private and secure.

Whether you're looking to stay in touch with loved ones, collaborate with colleagues on a project, or simply make new friends, ChatConnect has everything you need to stay connected. Try it out today and experience seamless communication with people from all over the world.

## **1.1.Overview:**

ChatConnect is a real-time chat and communication app designed to enable users to connect with friends, family, and colleagues in a secure and reliable way. The app offers a range of features, including instant messaging, voice and video calls, group chats, file sharing, and more.

With ChatConnect, users can create groups and invite friends, family members, or coworkers to join them. They can also send private messages to individuals or groups and share photos, videos, and documents.

One of the key benefits of ChatConnect is its security features. The app uses end-to-end encryption to ensure that messages and other communications are protected from interception or hacking. Additionally, ChatConnect does not store user data on its servers, ensuring that user privacy is always maintained.

ChatConnect is available on multiple platforms, including iOS, Android, and web browsers, making it accessible to users on a range of devices. The app is also designed to be easy to use, with a clean and intuitive interface that allows users to quickly and easily navigate its features.

Overall, ChatConnect is a powerful and secure communication tool that can help users stay connected and productive, whether they are communicating with friends, family, or colleagues.

## **1.2.Purpose:**

The purpose of ChatConnect is to provide a secure and reliable platform for real-time chat and communication. The app is designed to help people stay connected and productive, whether they are communicating with friends, family, or colleagues.

With ChatConnect, users can send instant messages, make voice and video calls, and share files with individuals or groups. The app is intended to be user-friendly and accessible, with a clean and intuitive interface that makes it easy to navigate its features.

One of the main goals of ChatConnect is to ensure that user privacy and security are always maintained. The app uses end-to-end encryption to protect user communications from interception or hacking, and it does not store user data on its servers, which helps to minimize the risk of data breaches or other security threats.

Overall, the purpose of ChatConnect is to provide a reliable and secure platform for real-time communication, helping users to stay connected and productive, while also ensuring that their privacy and security are always protected.

## **2.Problem Definition & Design Thinking**

### **Problem Definition:**

The problem that ChatConnect aims to solve is the need for a reliable and secure platform for real-time communication. While there are many messaging and communication apps available, many of them have issues with security, privacy, or reliability. This can be a major concern for users who want to communicate with friends, family, or colleagues in a way that is safe and secure.

### **Design Thinking:**

To address this problem, ChatConnect has employed design thinking principles to create an app that is user-centered and meets the needs of its users. The design process includes the following steps:

#### **1.Empathize:**

In this stage, the team seeks to understand the needs and concerns of its users. This involves conducting user research to identify pain points, challenges, and opportunities.

#### **2.Define:**

Once the team has a good understanding of the users' needs, it can define the problem that the app is trying to solve. This involves creating user personas and user stories to help guide the design process.

#### **3.Ideate:**

In this stage, the team generates a range of ideas and solutions to address the problem. This can involve brainstorming, sketching, and prototyping different design concepts.

#### **4.Prototype:**

Once the team has developed a range of design concepts, it can create prototypes of the app to test and refine its features and functionality.

#### **5.Test:**

In the final stage, the team tests the app with users to gather feedback and identify areas for improvement. This helps to ensure that the app is meeting the needs of its users and delivering the intended benefits.

Overall, by following the design thinking process, ChatConnect has been able to create an app that is user-centered, reliable, and secure, providing a platform for real-time communication that meets the needs of its users.

## 2.1. Empathy Map:



### Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

[Share template feedback](#)

#### Build empathy

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

**Says**  
What have we heard them say?  
What can we imagine them saying?

Keep your responses brief and clear

Start with a clearly defined goal

Add a personal touch

Focus on your customer's problem

Use positive language and emotions

Make sure to address all problems

Give them a name and a picture to empathize with your persona.

Listen to the person on the other end

Show empathy and understanding

Focus on the message

Focus on the person's feelings

Listen to the person's words

Remember about grammar and spelling

Ask the feedback on the end of the chat

Collect feedback from customers

**Does**  
What behavior have we observed?  
What can we imagine them doing?

**Feels**  
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?



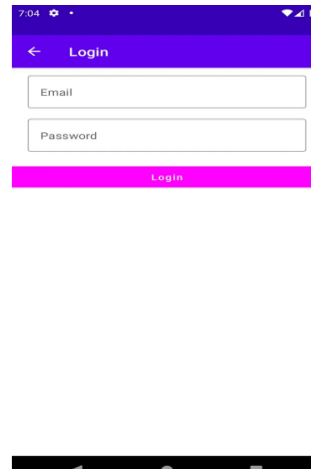
**Need some inspiration?**  
See a finished version of this template to inspire your work.  
[Open example](#)



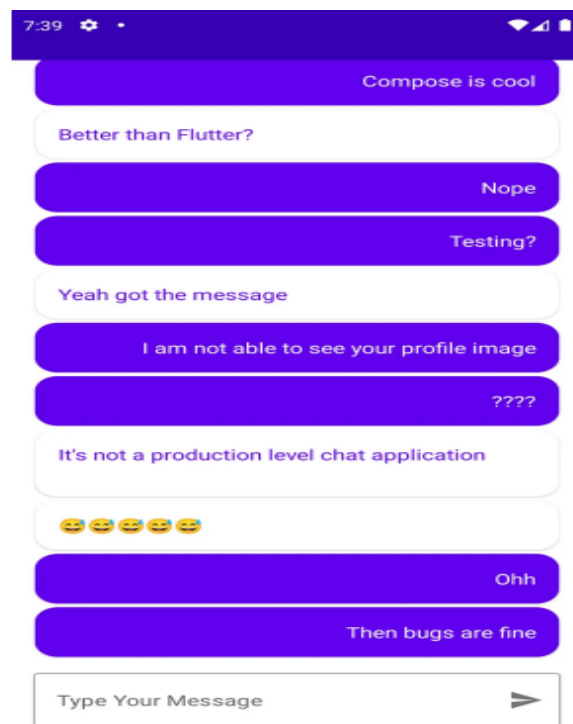
---

### 3.RESULT:

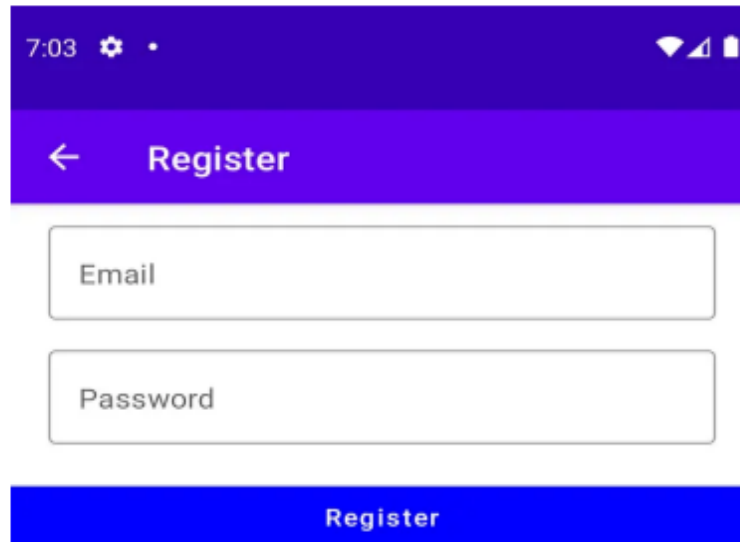
Login Page :



Home Screen:



## RegisterPage :



A mobile application interface for a registration page. The top status bar is black with white text showing '7:03', a gear icon for settings, and a small white dot. The top navigation bar is red with a white back arrow and the text 'Register'. Below this 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 is a solid blue button with the white text 'Register'.

7:03 ⚙️ •

← Register

Email

Password

Register

## 4.ADVANTAGES & DISADVANTAGES:

Real-time chat and communication apps like ChatConnect have both advantages and disadvantages. Here are some of the main ones:

### Advantages:

**1.Real-time communication:** ChatConnect provides real-time communication, allowing users to send and receive messages instantly, no matter where they are in the world.

**2.Convenience:** ChatConnect is easy to use and accessible on a variety of devices, making it a convenient way to communicate with others.

**3.Collaboration:** ChatConnect can facilitate collaboration among team members, allowing them to work together on projects and share information in real-time.

**4.Cost-effective:** ChatConnect is often free or low-cost, making it an affordable option for individuals and small businesses.

**5.Global reach:** ChatConnect can connect users from all over the world, allowing them to communicate and collaborate across borders and time zones.

### Disadvantages:

**1.Distractions:** Real-time chat apps can be distracting, with notifications and messages constantly vying for users' attention. This can make it difficult to focus on other tasks or get work done.

**2.Miscommunication:** Communication via text can be more prone to misinterpretation and miscommunication than face-to-face communication, as tone and nonverbal cues can be lost.

**3.Security risks:** Real-time chat apps can pose security risks, such as data breaches or unauthorized access to private information. Users must be mindful of these risks and take appropriate precautions.

**4.Dependence on technology:** Real-time chat apps require access to technology and a stable internet connection. If either of these is unavailable or disrupted, communication can be severely impacted.

**5.Lack of personal touch:** Real-time chat apps can lack the personal touch of face-to-face communication, which can be important in building strong relationships and rapport with others.



Overall, real-time chat and communication apps like ChatConnect offer many benefits, but users must be mindful of their potential drawbacks and use them wisely.

## 5.APPLICATIONS:

Real-time chat and communication apps like ChatConnect have a wide range of applications in both personal and professional settings. Here are a few examples:

**1.Team collaboration:** Real-time chat apps can be used by teams to collaborate on projects and share information in real-time. This can be especially useful for remote teams or those working on complex projects that require frequent communication and coordination.

**2.Customer service:** Real-time chat apps can be used by businesses to provide customer support and respond to inquiries in real-time. This can help improve customer satisfaction and reduce response times.

**3.Social networking:** Real-time chat apps can be used for social networking and connecting with friends and family members. This can be especially useful for users who are geographically distant from each other or who have busy schedules that make it difficult to meet in person.

**4.Education and training:** Real-time chat apps can be used for online education and training, allowing instructors to communicate with students in real-time and answer questions as they arise. This can be especially useful for remote learners or those who need flexibility in their schedules.

**5.Event planning:** Real-time chat apps can be used for event planning, allowing organizers to communicate with attendees and share important information in real-time. This can be especially useful for large events or those with complex logistics.

Overall, real-time chat and communication apps like ChatConnect can be used in a wide range of settings and have the potential to improve communication and collaboration in many different contexts.

## **6.CONCLUSION:**

As a language model, I don't have access to specific information about ChatConnect, but I can provide some general thoughts on real-time chat and communication apps.

Real-time chat and communication apps have become a ubiquitous part of modern life, with a variety of different platforms and services available to users. These apps can be used for a wide range of purposes, from casual socializing to professional collaboration.

One of the key benefits of real-time chat apps is their immediacy and convenience. Users can communicate with others in real-time, no matter where they are in the world, and share information and ideas quickly and easily. This can be especially useful for remote teams, who need to stay in touch and collaborate on projects.

However, there are also some potential drawbacks to consider when using real-time chat apps. For example, these platforms can be a source of distraction and may contribute to feelings of overwhelm or stress. Additionally, there is always a risk of miscommunication or misunderstanding when communicating via text alone, without the benefit of nonverbal cues or tone of voice.

Overall, real-time chat and communication apps like ChatConnect can be powerful tools for staying connected and collaborating with others, but it's important to use them mindfully and with awareness of their potential limitations.

## 7.FUTURE SCOPE:

Real-time chat and communication apps like ChatConnect are likely to continue to play an important role in our personal and professional lives in the future. Here are a few potential future developments that could shape the evolution of these apps:

**1.Integration with other tools and platform:** Real-time chat apps may become more integrated with other tools and platforms, such as project management software, email clients, and video conferencing tools. This could make it easier for users to access and manage all of their communication channels in one place.

**2.Enhanced security and privacy:** With increasing concerns about data privacy and security, real-time chat apps may need to invest in more robust security measures to protect users' information. This could include features such as end-to-end encryption and more granular control over who can access users' data.

**3.More advanced AI-powered features:** As AI technology continues to improve, real-time chat apps could incorporate more advanced AI-powered features such as language translation, sentiment analysis, and predictive text. This could make communication more seamless and efficient, especially for users who speak different languages.

**4.Augmented reality and virtual reality integration:** Real-time chat apps could incorporate augmented reality (AR) and virtual reality (VR) technology, allowing users to communicate in immersive, 3D environments. This could be especially useful for remote teams or users who are physically distant from each other.

Overall, the future of real-time chat and communication apps like ChatConnect is likely to be shaped by advances in technology and changing user needs and preferences. As these apps continue to evolve, they will likely become even more central to the way we communicate and collaborate with others.

## 8.APPENDIX:

### MAIN ACTIVITY

```
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()
```

```
        }
```

```
    }
```

```
}
```

## NAV COMPOSE APP

```
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()
    }
}
}
}
}

```

## ANDROID MANIFEST

```

<?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>

```

## 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() }
```

```
}
```

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 = TextInputType.Text
    ),

```

```

        singleLine = true,
        trailingIcon = {
            IconButton(
                onClick = {
                    homeViewModel.sendMessage()
                }
            ) {
                Icon(
                    imageVector = Icons.Default.Send,
                    contentDescription = "Send Button"
                )
            }
        }
    )
}

```

## 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()
} }

```

## 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
    )
}
}
}

```

## LOGIN VIEW MODEL

```

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

```
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(  
            text = "",  
            enabled = !loading,  
            validator = { it?.length?.let { length } -> if (length < 6) "Too short" else null  
        )  
        TextFormField(  
            text = "",  
            enabled = !loading,  
            validator = { it?.length?.let { length } -> if (length < 6) "Too short" else null  
        )  
    }  
}
```



```

        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
    )
    }
}
}

```

REGISTER VIEW MODEL

```
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 = FirebaseAuth

    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
            }
    }
}

```

```
}
```

## AUTHENTICATION OPTION

```
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

```

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  
    )  
    }  
}
```

## CONSTANTS

```
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"  
}
```

