INDEX

1. INTRODUCTION

Welcome to the introduction of our Chat Application project! Our team has created a chat application that aims to simplify communication between groups in a user-friendly and secure manner.

Our chat application has been designed to provide users with a seamless chatting experience across android platforms, including mobile devices. Whether you are at home, in the office, or on the go, you can use our chat application to stay connected with your friends, family, and colleagues.

Our application features a user-friendly interface that is easy to navigate, making it simple for users to find and use the features they need. We have included a range of features to enhance the user experience.

To ensure the security and privacy of our users, we have implemented several security measures, such as end-to-end encryption and authentication. This means that your conversations are protected from unauthorized access, ensuring that you can chat with confidence.

In addition, our chat application allows users to join groups, making it easy to collaborate with colleagues or stay in touch with friends and family

Overall, our Chat Application project aims to provide users with a reliable, user-friendly, and secure platform for communication. We are excited to share our application with you and hope that it meets all of your communication needs.

1.1. Overview

Our Chat Application project is a Android-based chat application that allows users to communicate with each other in real-time using text-based messages. The application is designed to be simple and user-friendly, with an intuitive interface that is easy to navigate.

The application allows users to easy to connect with friends, family, and colleagues. Users can create group chats, where multiple users can communicate with each other at the same time

The application includes several features to enhance the user experience, including the ability to send text messages. so they can stay up-to-date with their conversations even when they are not actively using the application.

To ensure the security and privacy of our users, the application uses end-to-end encryption to protect conversations from unauthorized access. Users can also enabled authentication for an additional layer of security.

The Chat Application project is developed using Jetpack Compose in Android Studio. The application is designed to be responsive and works seamlessly across different android devices

Overall, our Chat Application project is a reliable, user-friendly, and secure platform for communication, designed to meet the needs of individuals and groups who want to stay connected in real-time.

1.2. Purpose

The purpose of our Chat Application project is to provide a reliable, user-friendly, and secure platform for real-time communication. Our goal is to make it easy for groups to stay connected with each other, regardless of their location.

We believe that effective communication is essential for building strong relationships, collaborating on projects, and achieving common goals. With our Chat Application project, we aim to provide a platform that facilitates communication in a way that is convenient, intuitive, and secure.

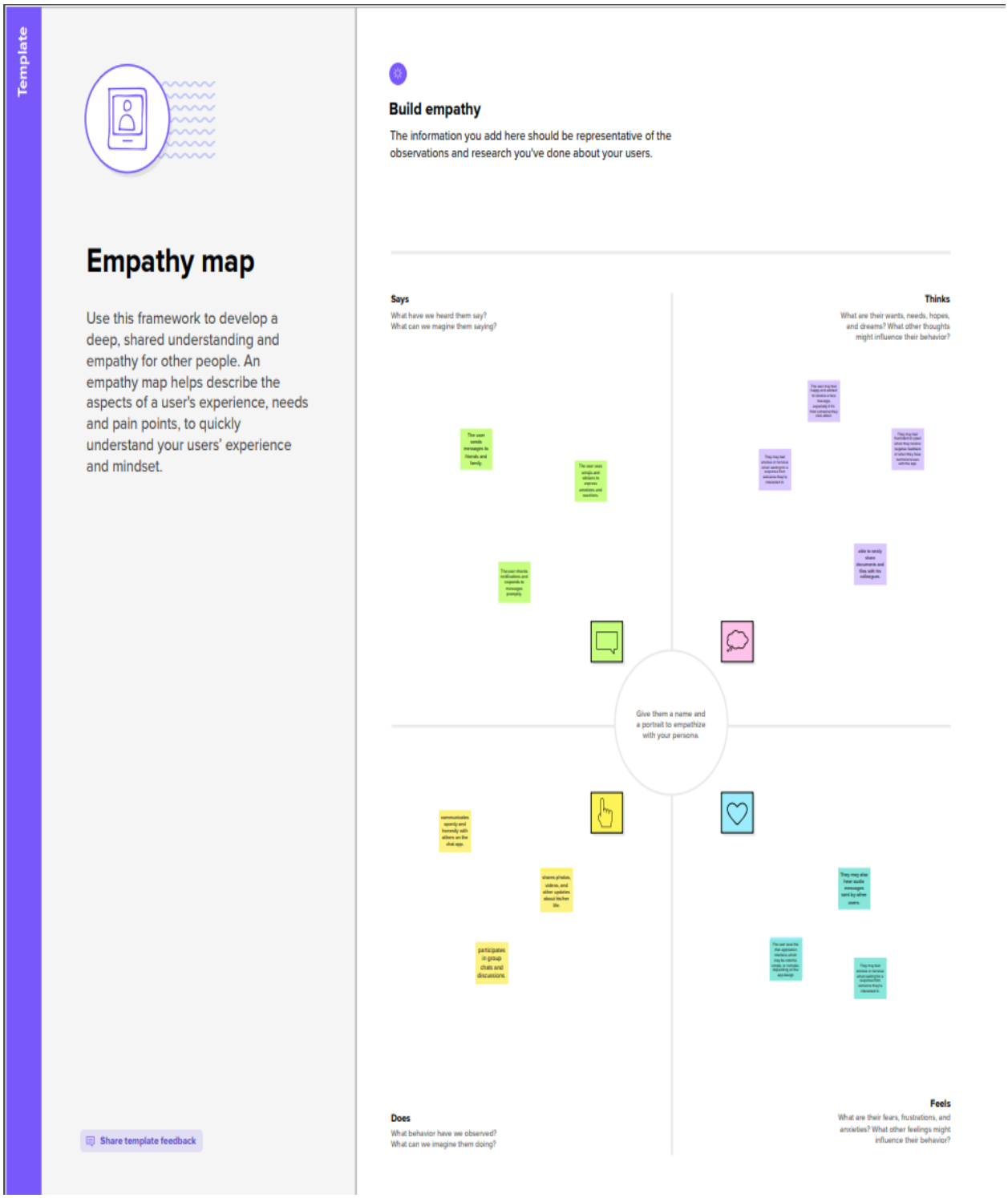
In particular, we want to provide a chat application that meets the needs of individuals and groups who value privacy and security. To achieve this, we have implemented features such as end-to-end encryption authentication to ensure that conversations are protected from unauthorized access.

Our Chat Application project is also designed to be user-friendly, with an intuitive interface that makes it easy for users to navigate and use the application. We believe that a user-friendly design is essential for ensuring that our application is accessible to a wide range of users, regardless of their technical expertise.

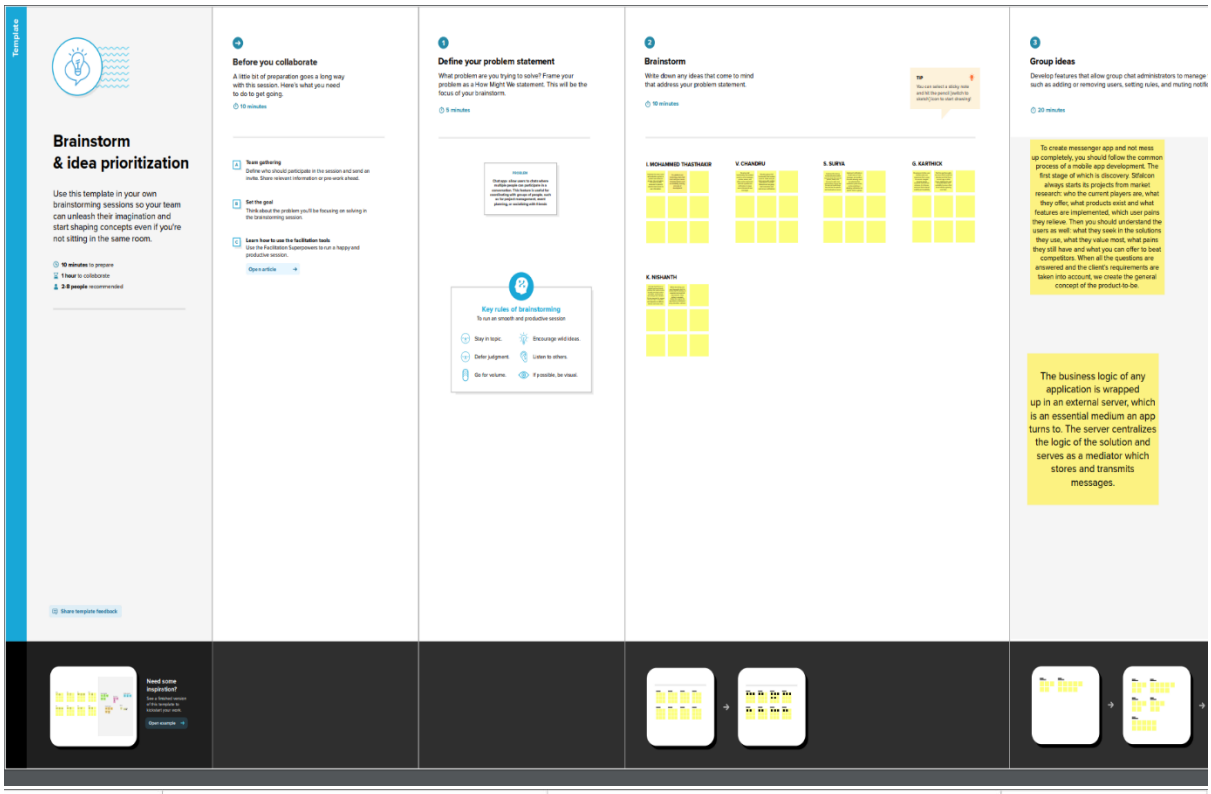
Overall, the purpose of our Chat Application project is to provide a reliable, user-friendly, and secure platform for real-time communication that meets the needs of groups who value privacy and convenience. We hope that our application will help users stay connected and communicate effectively with each other, whether they are friends, family members, or colleagues.

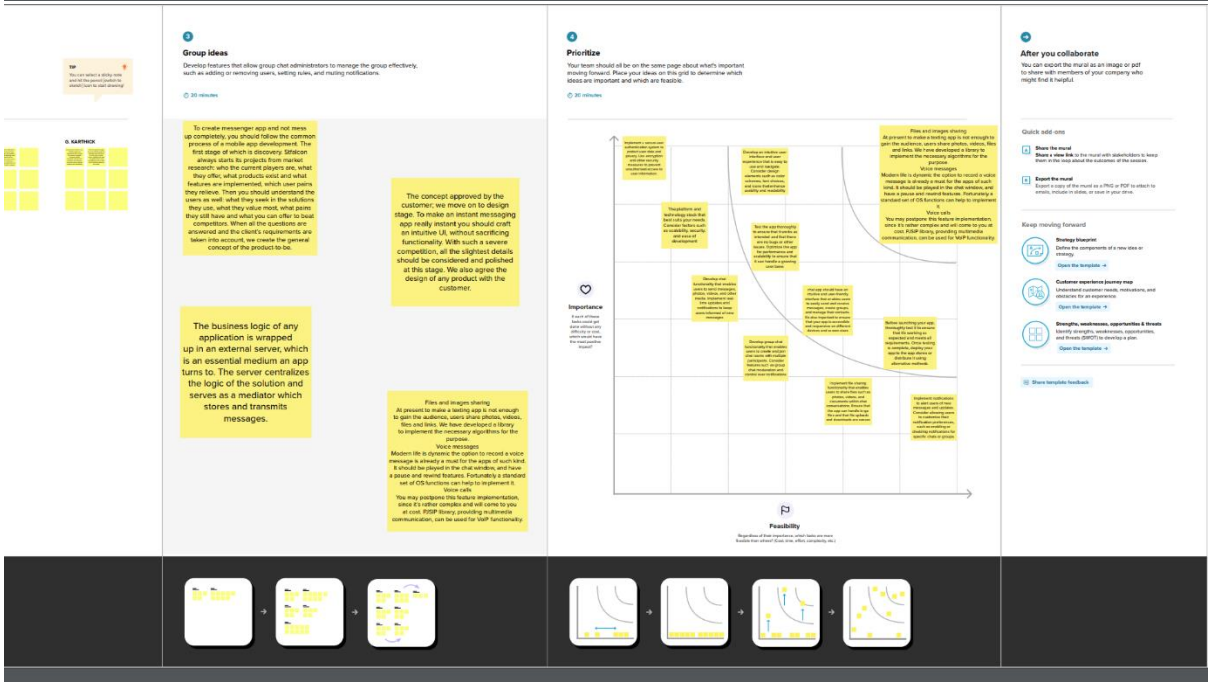
2. PROJECT DEFINITION AND DESIGN THINKING

2.1. Empathy Map

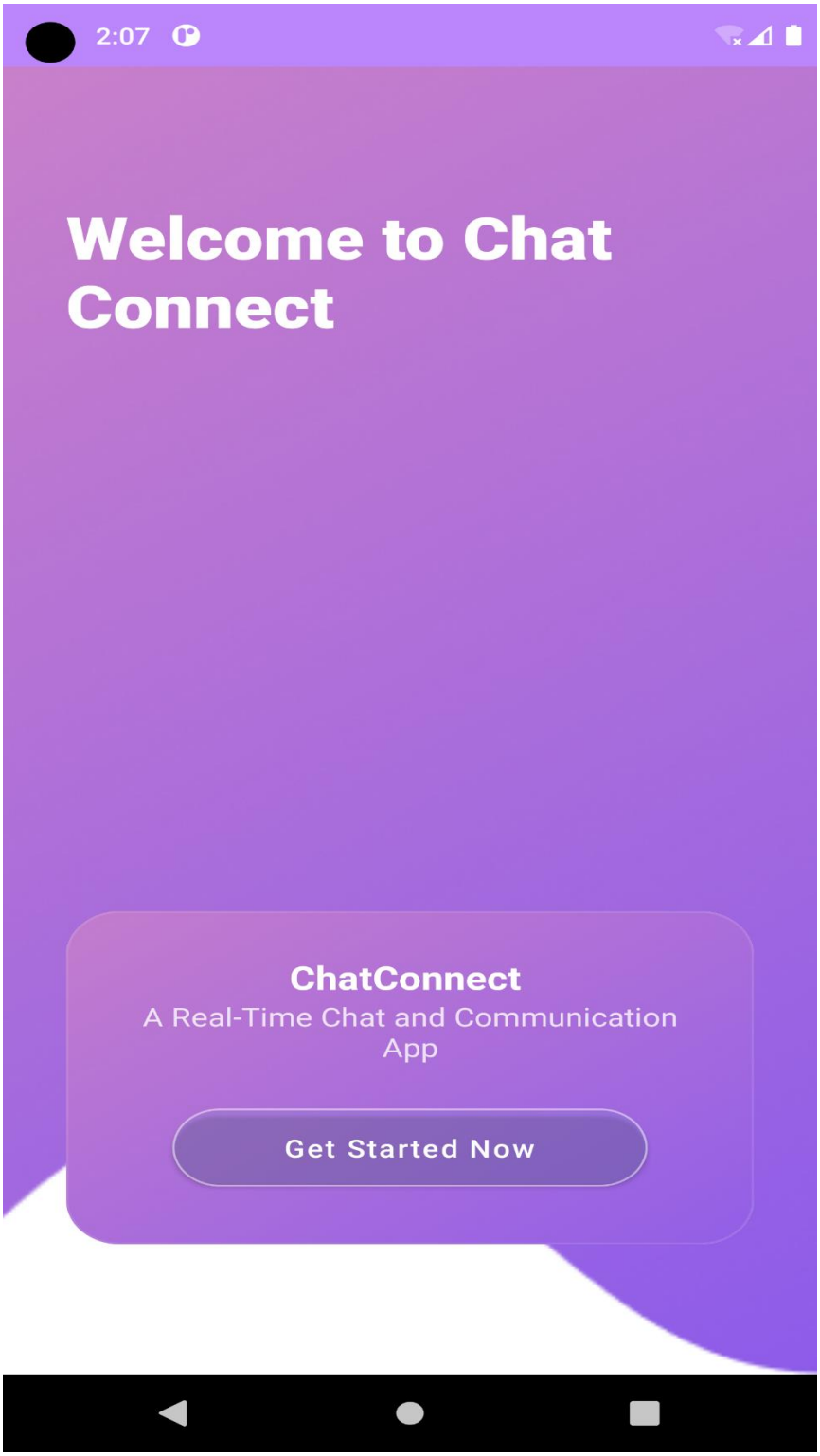


2.2. Ideation & Brainstorming Map

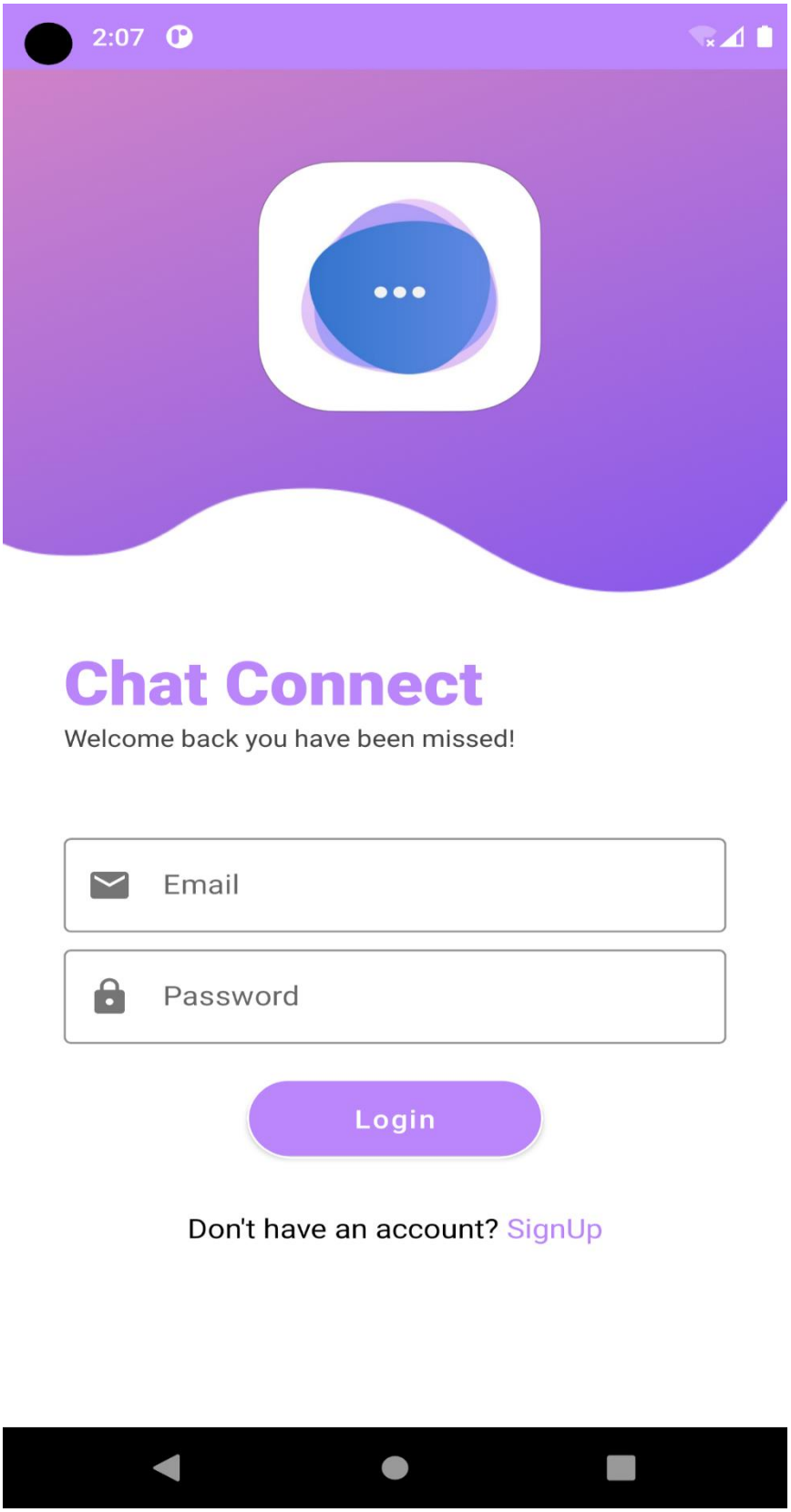




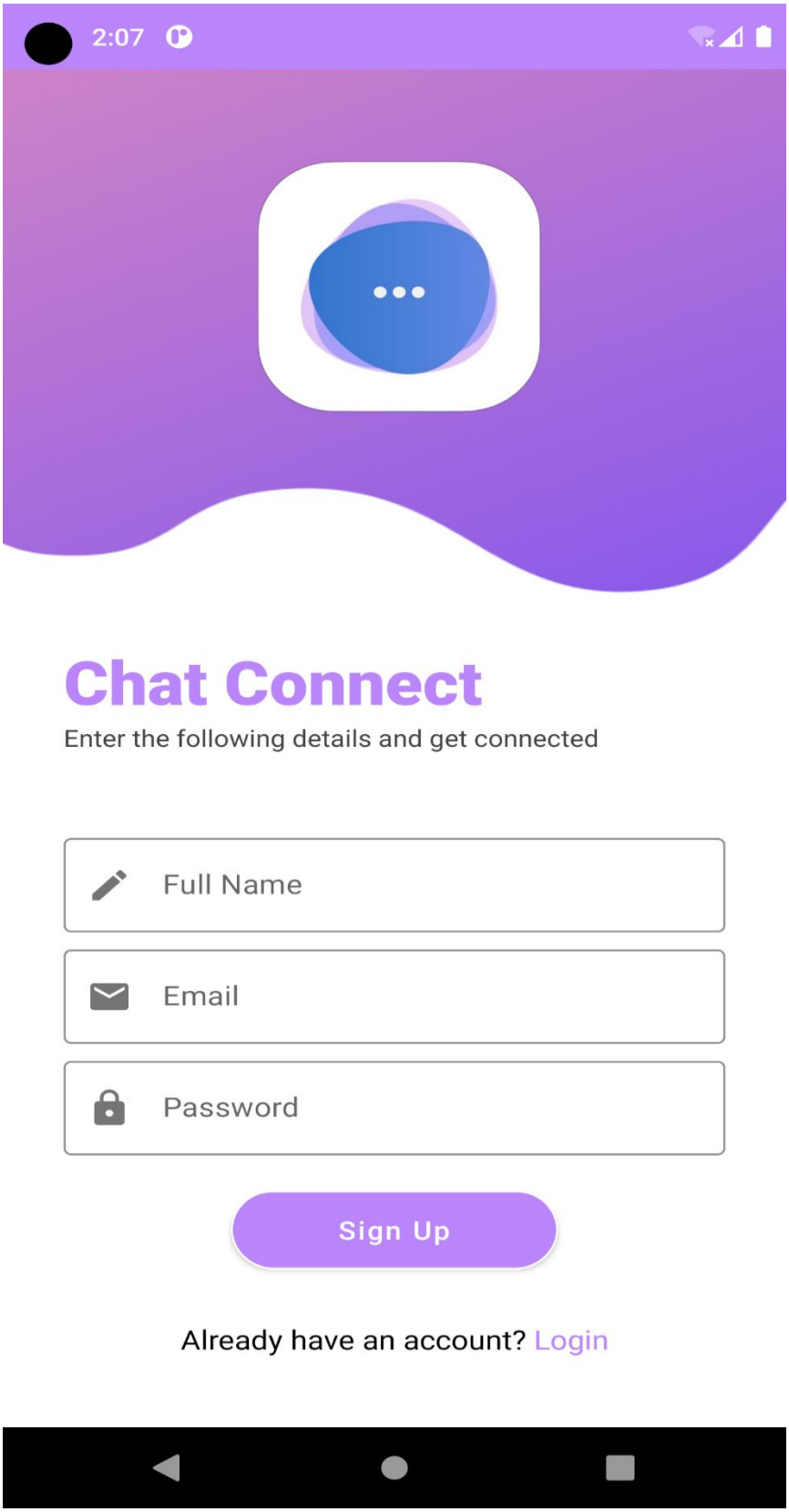
3. RESULT



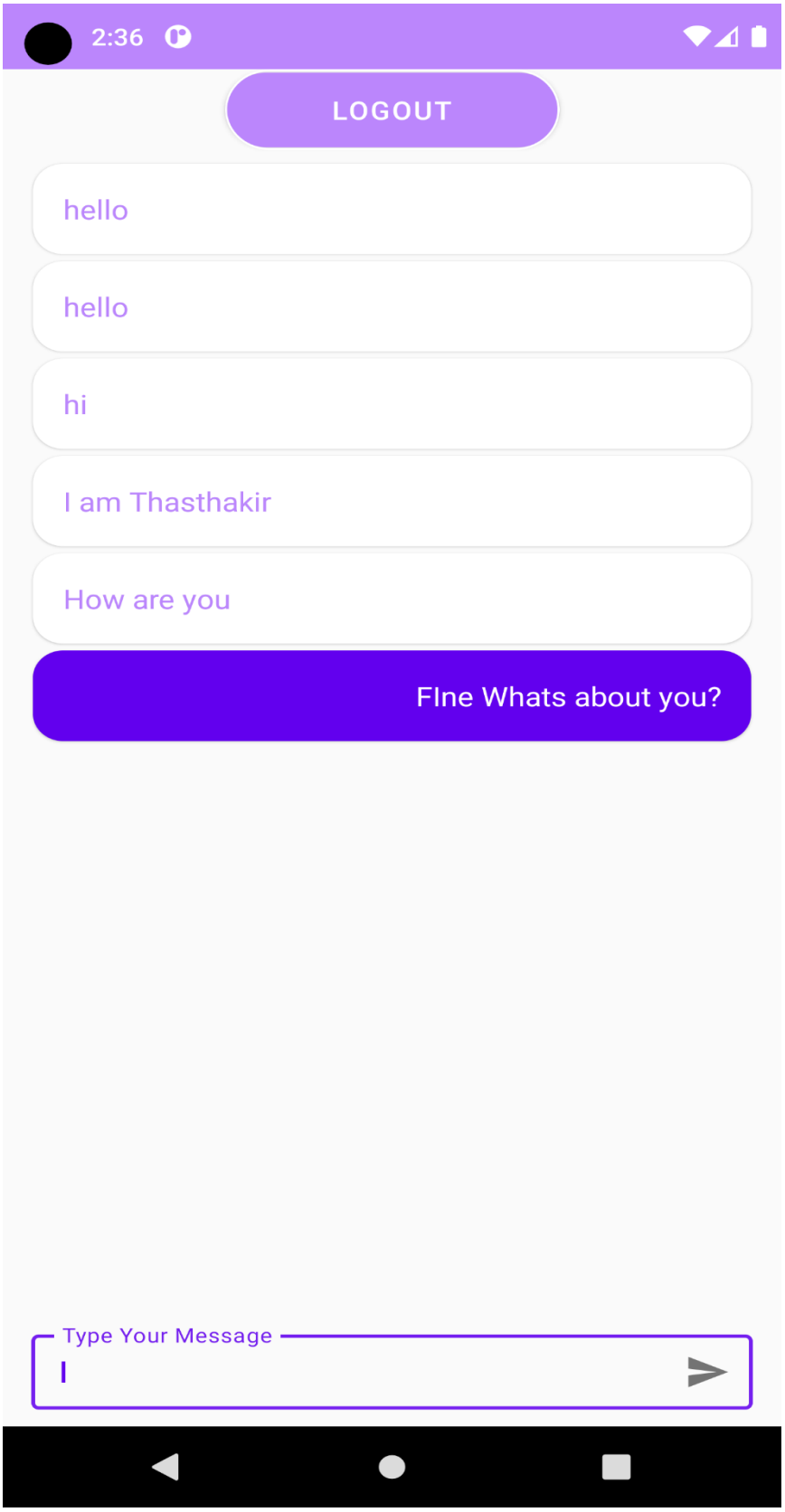
3.1. Authentication Option Activity



3.2. Login Activity



3.3. Register Activity



3.4. Main Activity

4. ADVANTAGES & DISADVANTAGES

ADVANTAGES

Real-time communication: Chat applications provide real-time communication, allowing users to communicate with each other instantly. This is especially useful for individuals and groups who need to collaborate or coordinate quickly.

Convenience: Chat applications can be accessed from anywhere with an internet connection, making them a convenient way to communicate with others. Users can communicate with each other regardless of their location, making it easier to stay connected.

Cost-effective: Chat applications are often free to use, making them a cost-effective way to communicate with others. This is particularly useful for groups who need to communicate frequently, as it can save on long-distance or international calling charges.

Group communication: Chat applications allow users to communicate in groups, making it easier to collaborate on projects or coordinate with multiple people. This is particularly useful for teams working on a project or for families and friends who want to stay in touch.

Security: Chat applications can provide secure communication through features such as end-to-end encryption and authentication. This helps to ensure that conversations are protected from unauthorized access.

Overall, chat applications provide a convenient and cost-effective way to communicate with others in real-time, with added security features to ensure privacy and protect conversations. They are an efficient way to collaborate and stay connected with others regardless of location.

DISADVANTAGES

Misinterpretation of messages: Text-based communication can be prone to misinterpretation as it lacks the nuance of tone of voice and body language. This can lead to misunderstandings or miscommunications, particularly in sensitive or emotionally charged conversations.

Distraction: The constant notifications and alerts from chat applications can be distracting and disrupt productivity, particularly in work environments where focus and concentration are important.

Information overload: Chat applications can result in information overload, particularly in group chats or in busy chat environments. Users may struggle to keep up with the volume of messages or may miss important information.

Dependence on technology: Chat applications are dependent on technology, which can be unreliable at times. Technical issues or internet outages can disrupt communication and make it difficult to stay connected with others.

Security risks: While chat applications can provide secure communication, there is always a risk of data breaches or hacking, particularly if users do not take appropriate precautions, such as using strong passwords and enabling two-factor authentication.

Overall, chat applications can be useful tools for communication, but they also have some potential disadvantages that users should be aware of. It's important to use chat applications in a thoughtful and responsible way, and to balance the convenience of real-time communication with the potential risks and distractions.

5. APPLICATIONS

Personal communication: Chat applications are commonly used for personal communication between friends and family members. They can be used to exchange messages, share photos and videos, and keep in touch with loved ones who live far away.

Business communication: Chat applications are also widely used for business communication. They can be used to communicate with colleagues, clients, and customers in real-time. Many chat applications also offer features such as video conferencing, screen sharing, and document collaboration, which can be helpful for remote teams and distributed workforces.

Customer support: Chat applications are increasingly being used by businesses to provide customer support. They allow customers to communicate with support representatives in real-time and can be more convenient and efficient than traditional support channels such as phone and email.

Education: Chat applications can also be used in education settings to facilitate communication between teachers and students. They can be used to answer questions, provide feedback, and facilitate group discussions.

Social networking: Some chat applications are designed specifically for social networking and can be used to meet new people, join interest groups, and share information and ideas.

Overall, chat applications can be used in a wide range of contexts and for various purposes, making them a versatile and useful communication tool.

6. CONCLUSION

In conclusion, our Chat Application project aims to provide a reliable, user-friendly, and secure platform for real-time communication that meets the needs of individuals and groups who value privacy and convenience. The advantages of using a chat application for communication include real-time communication, convenience, cost-effectiveness, group communication, and security. However, there are also potential disadvantages such as misinterpretation of messages, distraction, information overload, dependence on technology, and security risks that users should be aware of.

We have designed our Chat Application project to address these potential disadvantages through features such as a user-friendly interface, end-to-end encryption, and two-factor authentication. We believe that our application will help users stay connected and communicate effectively with each other, whether they are friends, family members, or colleagues.

We hope that our Chat Application project will provide a useful and valuable tool for individuals and groups who need to communicate in real-time, and we are excited to continue developing and improving our application to meet the needs of our users.

7. FUTURE SCOPE

The Chat Application project has a lot of potential for future development and expansion. Some potential areas for future scope include:

Voice and video calling: While text-based communication is useful, some users may prefer to communicate through voice or video calls. Adding voice and video calling features could help to improve the user experience and make the application more versatile.

Integration with other applications: The Chat Application project could be integrated with other applications, such as email, social media, or project management tools. This would help to streamline communication and make it easier for users to stay connected across different platforms.

Advanced search and filtering: As the volume of messages in a chat application grows, it can become difficult to find specific information. Adding advanced search and filtering features could help users to quickly find the messages and information they need.

Translation and localization: Chat applications are used by people all over the world, and adding translation and localization features could help to improve accessibility and expand the user base.

Artificial intelligence and machine learning: Incorporating artificial intelligence and machine learning technologies could help to improve the user experience and make the application more intuitive and personalized. For example, the application could suggest responses or provide automatic translations based on user preferences and usage patterns.

8. APPENDIX

A. Source Code

MainActivity.kt

```
package com.example.chatapp
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.FirebaseApp
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        FirebaseApp.initializeApp(this)
        setContent {
            NavComposeApp()
        }
    }
}
```

NavComposeApp.kt

```
package com.example.chatapp
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.example.chatapp.nav.Action
```

```
import com.example.chatapp.nav.Destination.AuthenticationOption
import com.example.chatapp.nav.Destination.Home
import com.example.chatapp.nav.Destination.Login
import com.example.chatapp.nav.Destination.Register
import com.google.firebase.auth.FirebaseAuth
import com.example.chatapp.ui.theme.ChatAppTheme
import com.example.chatapp.view.AuthenticationView
import com.example.chatapp.view.home.HomeView
import com.example.chatapp.view.login.LoginView
import com.example.chatapp.view.register.RegisterView

@Composable
fun NavComposeApp() {
    val navController = rememberNavController()
    val actions = remember(navController) { Action(navController) }
    ChatAppTheme {
        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,
                    login = actions.login
                )
            }
            composable(Login) {
                LoginView(
                    home = actions.home,
                    back = actions.navigateBack,
                    register = actions.register
                )
            }
            composable(Home) {
                HomeView()
            }
        }
    }
}
```

Navigation.kt

```
package com.example.chatapp.nav
import androidx.navigation.NavHostController
import com.example.chatapp.nav.Destination.Home
import com.example.chatapp.nav.Destination.Login
import com.example.chatapp.nav.Destination.Register

object Destination {
    const val AuthenticationOption = "authenticationOption"
    const val Register = "register"
    const val Login = "login"
    const val Home = "home"
}

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() }
}
```

Constants.kt

```
package com.example.chatapp
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"
}
```

AuthenticationOption.kt

```
package com.example.chatapp.view
import androidx.compose.foundation.Image
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.R
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
```

```
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.chatapp.view.ui.theme.ChatAppTheme
@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {
    ChatAppTheme {

        Surface(
            color = Color.White
        ) {
            Box {
                Image(painter = painterResource(id = com.example.chatapp.R.drawable.bg_login),
                    contentDescription ="bg", modifier = Modifier.fillMaxSize(),
                    contentScale = ContentScale.Crop)
                Column(
                    modifier = Modifier
                        .padding(
                            horizontal = 32.dp, vertical = 80.dp
                        )
                        .fillMaxSize()
                ) {
                    Text(text = "Welcome to Chat Connect", color = Color.White,
                        fontWeight = FontWeight.Black,
                        fontSize = 36.sp
                    )
                    Spacer(modifier = Modifier.fillMaxHeight(0.63f))
                    Card(
                        elevation = 4.dp,
                        modifier = Modifier
```

```
            .border(
                width = 1.dp,
                color = Color.White.copy(0.1f),
                shape = RoundedCornerShape(27.dp)
            )
            .clip(RoundedCornerShape(27.dp))
        ) {
        Image(
            painter = painterResource(id = com.example.chatapp.R.drawable.bg_login),
            contentDescription ="Card_Bg" ,
            modifier = Modifier.fillMaxSize(),
            contentScale = ContentScale.Crop
        )
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier
                .padding(27.dp)
                .fillMaxSize()
        ) {
            Text(
                text = "ChatConnect ",
                color = Color.White,
                fontWeight = FontWeight.Bold,
                fontSize = 20.sp,
                textAlign = TextAlign.Center
            )
            Text(text = "A Real-Time Chat and Communication App",
                color = Color.White.copy(0.8f),
                textAlign = TextAlign.Center
```

```
                )
                Spacer(modifier = Modifier.padding(bottom = 27.dp))
                Button(onClick = login, shape = RoundedCornerShape(percent = 50),
                    modifier = Modifier.border(width = 1.dp,
                        color = Color.White.copy(0.5f),
                        shape = RoundedCornerShape(percent = 50)
                    ), colors = ButtonDefaults.buttonColors(
                        backgroundColor = Color(151,169,246, alpha = 0x32),
                        contentColor = Color.White)) {
                    Text(text = "Get Started Now", modifier = Modifier.padding(horizontal =
40.dp, vertical = 4.dp),
                        fontSize = 15.sp,
                        fontWeight = FontWeight.SemiBold
                    )
                }
            }
        }
    }
}
}
}
```

Widgets.kt

```
package com.example.chatapp.view
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.example.chatapp.view.ui.theme.Purple200
@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) Purple200 else Color.White
        )
    }}
```

LoginViewModel.kt

```
package com.example.chatapp.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

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
                }
        }
    }
}
```

Login.kt

```
package com.example.chatapp.view.login
import androidx.compose.foundation.Image
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.layout.R
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.ClickableText
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Email
import androidx.compose.material.icons.filled.Lock
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.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel

@Composable
fun LoginView(
    home: () -> Unit,
```

```
    back: () -> Unit,
    register: () -> 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)
    var text ="SignUp"
    Surface(
        color = Color.White
    ) {
        Box{
            Image(painter = painterResource(id = com.example.chatapp.R.drawable.bg_login),
                contentDescription ="bg",
                contentScale = ContentScale.FillBounds,
                modifier = Modifier
                    .fillMaxWidth()
                    .size(310.dp))

            Column(
                modifier = Modifier
                    .padding(
                        horizontal = 32.dp, vertical = 40.dp,
                    )
                    .fillMaxSize()
            ) {
                val Purple200 = Color(0xFFBB86FC)
                Image(painter = painterResource(id =
com.example.chatapp.R.drawable.app_logo_2),
                    contentDescription ="app_logo",
                    modifier = Modifier
```

```
            .fillMaxWidth()
            .size(180.dp))
        Spacer(modifier = Modifier.padding(bottom = 70.dp))
        Text(text = "Hello Again!", color = Color.White,
            fontWeight = FontWeight.Black,
            fontSize = 36.sp
        )
        Text(text = "Chat Connect", color = Purple200 ,
            fontWeight = FontWeight.Black,
            fontSize = 36.sp
        )
        Text(text = "Welcome back you have been missed!",
            color = Color.DarkGray,
//              fontWeight = FontWeight.Black,
            fontSize = 14.sp
        )
        Spacer(modifier = Modifier.padding(bottom = 50.dp))

        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier
                .padding(0.dp)
                .fillMaxSize()
        ) {
            val Purple200 = Color(0xFFBB86FC)
            OutlinedTextField(
                value = email,
                onValueChange = { loginViewModel.updateEmail(it) },

                placeholder = { Text(text = "Email") },
```

```
            colors = TextFieldDefaults.textFieldColors(
                backgroundColor = Color.White,
            ),
            leadingIcon = { Icon(imageVector = Icons.Default.Email, contentDescription
= "emailIcon") },
            modifier = Modifier
                .padding(bottom = 10.dp)
                .fillMaxWidth()

        )
        OutlinedTextField(
            value = password,
            onValueChange = { loginViewModel.updatePassword(it) },
            placeholder = { Text(text = "Password") },
            leadingIcon = { Icon(imageVector = Icons.Default.Lock, contentDescription =
"emailIcon") },

            colors = TextFieldDefaults.textFieldColors(
                backgroundColor = Color.White,
            ),
            modifier = Modifier
                .padding(bottom = 20.dp)
                .fillMaxWidth()
        )
        Button(onClick = { loginViewModel.loginUser(home = home) },

            shape = RoundedCornerShape(percent = 50),
            modifier = Modifier.border(width = 1.dp,
                color = Color.White,
                shape = RoundedCornerShape(percent = 50),
                ),
```

```
            colors = ButtonDefaults.buttonColors(
                backgroundColor = Purple200,
                contentColor = Color.White)) {
            Text(text = "Login",
                modifier = Modifier.padding(horizontal = 40.dp, vertical = 4.dp),
                fontSize = 15.sp,
                fontWeight = FontWeight.SemiBold
            )
        }
        if (loading) {
            CircularProgressIndicator()
        }
        Row(
            modifier = Modifier.padding(top = 30.dp)
        ) {
            Text(text = "Don't have an account? ", color = Color.Black)
            Text(
                modifier = Modifier
                    .clickable(enabled = true) {
                        register()
                    },
                text = text,
                color = Purple200,
            )
        }

    }

}
}}}
```

RegisterViewModel.kt

```
package com.example.chatapp.view.register
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.chatapp.Constants
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException

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 _name = MutableLiveData("")
    val name: LiveData<String> = _name

    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
    }
    // Update name
    fun updateName(newName: String) {
        _name.value = newName
    }
    // 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) {
                        val add = HashMap<String,Any>()
                        add["FullName"] = name
                        add["uid"] = Firebase.auth.currentUser?.uid.toString()
                        add["email"] = email
                        add["password"] = password
                        Firebase.firestore.collection("users").document().set(
                        add
                        ).addOnSuccessListener {
                            home()
                            _loading.value = false
                        }}              }       }    }}
```

Register.kt

```
package com.example.chatapp.view.register

import androidx.compose.foundation.Image
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.layout.R
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Close
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material.icons.filled.Email
import androidx.compose.material.icons.filled.Lock
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.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
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.compose.ui.unit.sp
```

```
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.chatapp.view.Appbar
import com.example.chatapp.view.Buttons
import com.example.chatapp.view.TextFormField

import com.example.chatapp.view.register.ui.theme.ChatAppTheme

@Composable
fun RegisterView(
    home: () -> Unit,
    back: () -> Unit,
    login: () -> Unit,
    registerViewModel: RegisterViewModel = viewModel()
) {
    val email: String by registerViewModel.email.observeAsState("")
    val loading: Boolean by registerViewModel.loading.observeAsState(initial = false)
    val password: String by registerViewModel.password.observeAsState("")
    val name: String by registerViewModel.name.observeAsState("")
    val text ="Login"

    Surface(
        color = Color.White
    ) {
        Box{
            Image(painter = painterResource(id = com.example.chatapp.R.drawable.bg_login),
                contentDescription ="bg",
                contentScale = ContentScale.FillBounds,
                modifier = Modifier
                    .fillMaxWidth()
                    .size(310.dp))
```

```
        Column(
            modifier = Modifier
                .padding(
                    horizontal = 32.dp, vertical = 40.dp,
                )
                .fillMaxSize()
        ) {
            val Purple200 = Color(0xFFBB86FC)
            Image(painter = painterResource(id =
com.example.chatapp.R.drawable.app_logo_2),
                contentDescription ="app_logo",
                modifier = Modifier
                    .fillMaxWidth()
                    .size(180.dp))
            Spacer(modifier = Modifier.padding(bottom = 70.dp))
            Text(text = "Welcome to", color = Color.White,
                fontWeight = FontWeight.Black,
                fontSize = 36.sp
            )
            Text(text = "Chat Connect", color = Purple200 ,
                fontWeight = FontWeight.Black,
                fontSize = 36.sp
            )
            Text(text = "Enter the following details and get connected",
                color = Color.DarkGray,
//                fontWeight = FontWeight.Black,
                fontSize = 14.sp
            )

            Spacer(modifier = Modifier.padding(bottom = 50.dp))
```

```
//
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                modifier = Modifier
                    .padding(0.dp)
                    .fillMaxSize()
            ) {
                val Purple200 = Color(0xFFBB86FC)

                OutlinedTextField(
                    value = name,
                    onValueChange = { registerViewModel.updateName(it) },

                    placeholder = { Text(text = "Full Name") },
                    colors = TextFieldDefaults.textFieldColors(
                        backgroundColor = Color.White,
                    ),
                    leadingIcon = { Icon(imageVector = Icons.Default.Edit, contentDescription =
"emailIcon") },
                    modifier = Modifier
                        .padding(bottom = 10.dp)
                        .fillMaxWidth()

                )
                OutlinedTextField(
                    value = email,
                   onValueChange = { registerViewModel.updateEmail(it) },

                    placeholder = { Text(text = "Email") },
                    colors = TextFieldDefaults.textFieldColors(
```

```
                backgroundColor = Color.White,
            ),
            leadingIcon = { Icon(imageVector = Icons.Default.Email, contentDescription
= "emailIcon") },
            modifier = Modifier
                .padding(bottom = 10.dp)
                .fillMaxWidth()

        )
        OutlinedTextField(
            value = password,
            onValueChange = { registerViewModel.updatePassword(it) },
            placeholder = { Text(text = "Password") },
            leadingIcon = { Icon(imageVector = Icons.Default.Lock, contentDescription =
"emailIcon") },

            colors = TextFieldDefaults.textFieldColors(
                backgroundColor = Color.White,
            ),
            modifier = Modifier
                .padding(bottom = 20.dp)
                .fillMaxWidth()

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

            shape = RoundedCornerShape(percent = 50),
            modifier = Modifier.border(width = 1.dp,
                color = Color.White,
                shape = RoundedCornerShape(percent = 50),
                ),
```

```
                colors = ButtonDefaults.buttonColors(
                    backgroundColor = Purple200,
                    contentColor = Color.White)) {
                Text(text = "Sign Up",
                    modifier = Modifier.padding(horizontal = 40.dp, vertical = 4.dp),
                    fontSize = 15.sp,
                    fontWeight = FontWeight.SemiBold
                )
            }
            if (loading) { CircularProgressIndicator()
            }
            Row(
                modifier = Modifier.padding(top = 30.dp)
            ) {
                Text(text = "Already have an account? ", color = Color.Black)
                Text(
                    modifier = Modifier
                        .clickable(enabled = true) {
                            login()
                        },
                    text = text,
                    color = Purple200,
                )
            }
        }
//          }
    }
  }
 }
}
```

HomeViewModel.kt

```
package com.example.chatapp.view.home
import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.chatapp.Constants
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException

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()
    }

    fun logout() {
    FirebaseAuth.getInstance().signOut()
    }
}
```

Home.kt

```
package com.example.chatapp.view.home

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
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.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.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.chatapp.Constants
import com.example.chatapp.view.SingleMessage
import com.example.chatapp.view.home.ui.theme.ChatAppTheme
import com.example.chatapp.view.home.ui.theme.Purple200
```

```
import com.google.firebase.auth.FirebaseAuth



@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
    ) {
        Button(onClick = { FirebaseAuth.getInstance().signOut() },

            shape = RoundedCornerShape(percent = 50),
            modifier = Modifier.border(width = 1.dp,
                color = Color.White,
                shape = RoundedCornerShape(percent = 50),

                ),



            colors = ButtonDefaults.buttonColors(
                backgroundColor = Purple200,
                contentColor = Color.White)) {
```

```
    Text(text = "LOGOUT",
        modifier = Modifier.padding(horizontal = 40.dp, vertical = 4.dp),
        fontSize = 15.sp,
        fontWeight = FontWeight.SemiBold
    )
}
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 = 10.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"
                )
            }
        }
    )
  }
}
```