PROJECT REPORT

CHATCONNECT

A Real-Time Chat and Communication App

INTRODUCTION

1.1 Overview

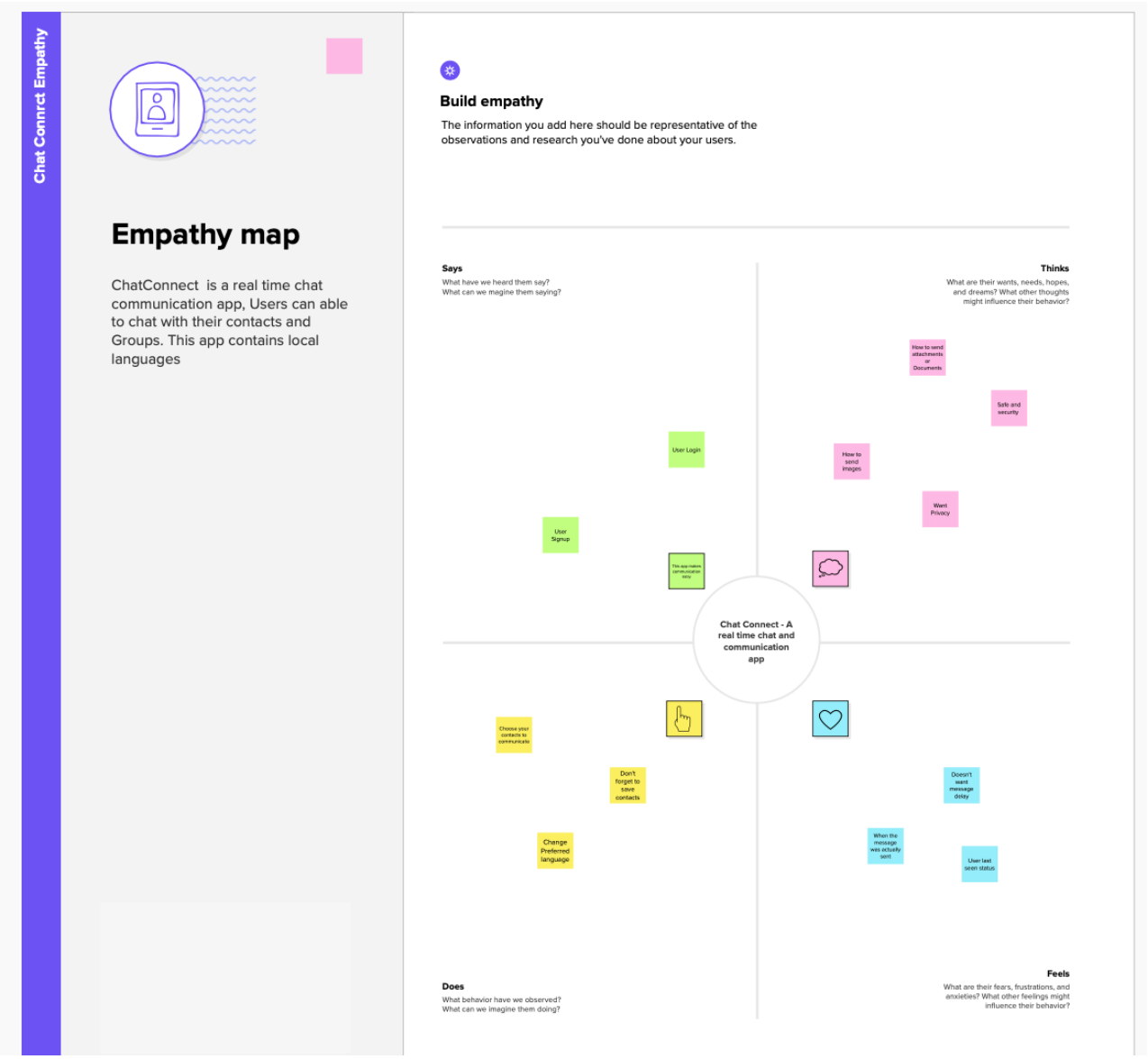
ChatConnect is a sample project built using the Android Compose UI toolkit. It demonstrates how to create a simple chat app using the Compose libraries. The app allows users to send and receive text messages. The project showcases the use of Compose's declarative UI and state management capabilities. It also includes examples of how to handle input and navigation using composable functions and how to use data from a firebase to populate the UI.

1.2 Purpose

The purpose of ChatConnect, a simple chat app, is to provide a convenient platform for people to communicate with each other in real-time. The app allows users to create an account and start chatting with individuals or groups. ChatConnect can be used for a variety of purposes, including staying in touch with friends and family, collaborating with colleagues, or meeting new people with similar interests. The app can be accessed from a desktop or mobile device, making it easy to stay connected on the go. Overall, ChatConnect aims to facilitate seamless and enjoyable communication between people.

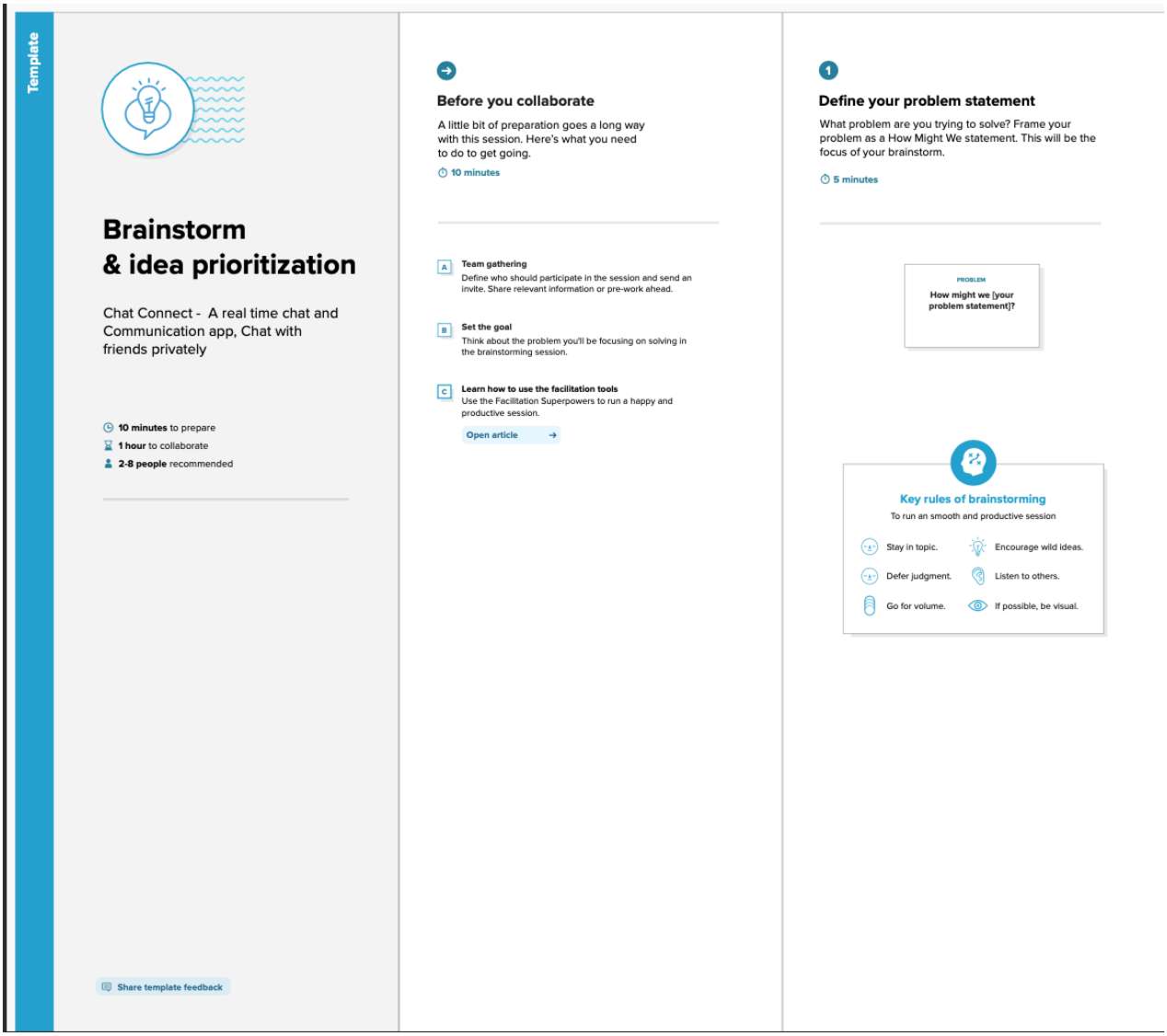
PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map



Empathy Map https://github.com/karthikeyan9952/chatconnect/documentations

2.2 Ideation & Brainstorming Map



Template

Brainstorm & idea prioritization

Chat Connect - A real time chat and Communication app, Chat with friends privately

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Share template feedback

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM
How might we [your problem statement]?

Key rules of brainstorming
To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Karthikeyan

- User Signup
- Navigation
- Handling HTTP Requests
- Localization
- User Last seen status
- Handling Project Folder Structure
- Publishing App

Monika

- User login
- Handling Navigation Graph
- JSON Decoding
- Change Preferred Language
- User Security
- Marketting

Vignesh

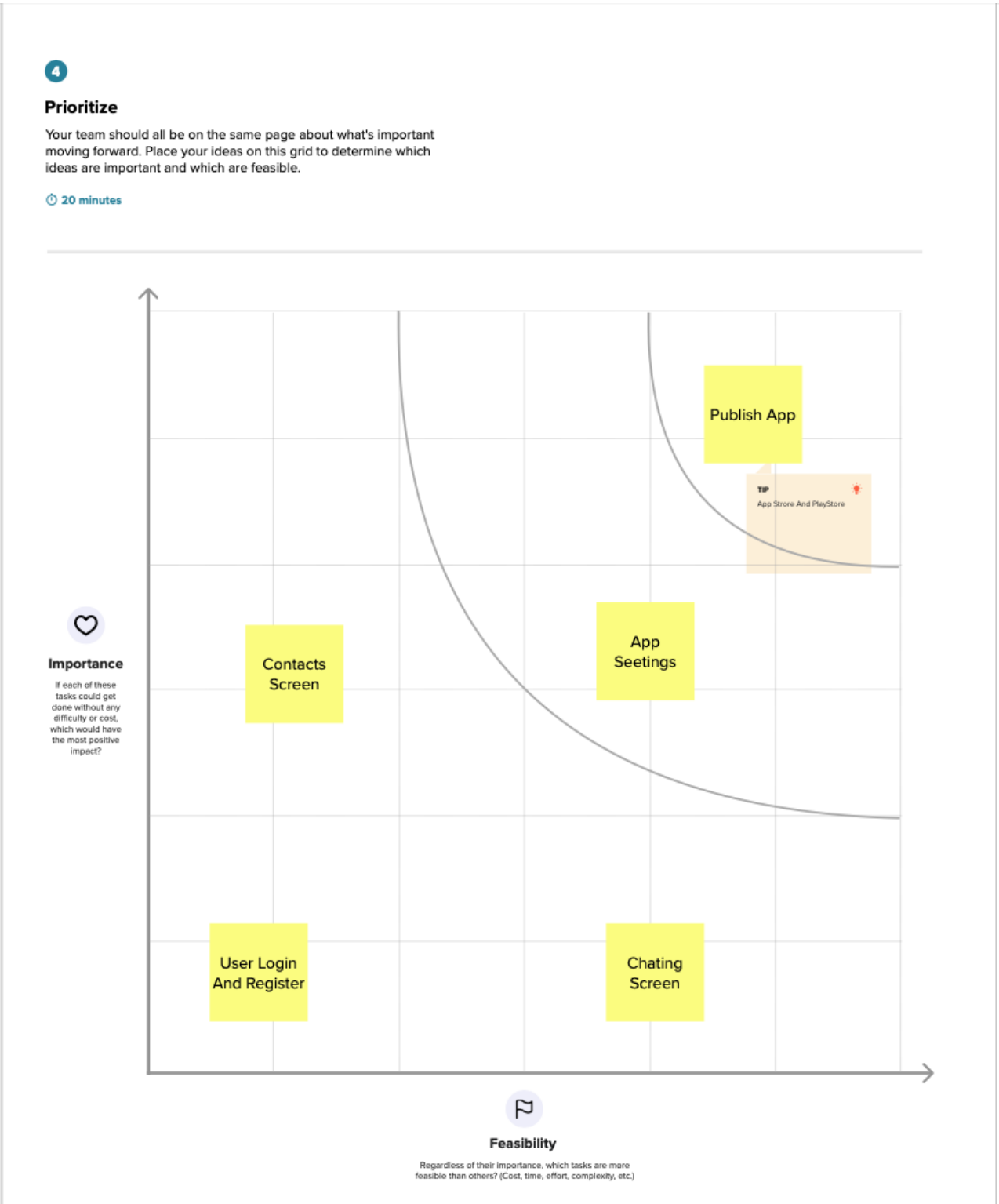
- Contact List
- State management
- JSON to Objects
- Sending Files
- Data Privacy
- Advertising

Shanmuganathan

- Contact Details
- Persisting Data Between Screens
- Passing Data to Compose View
- Attachments
- Sending UNICODE chars
- User Data Saving

MohanaPrasanth

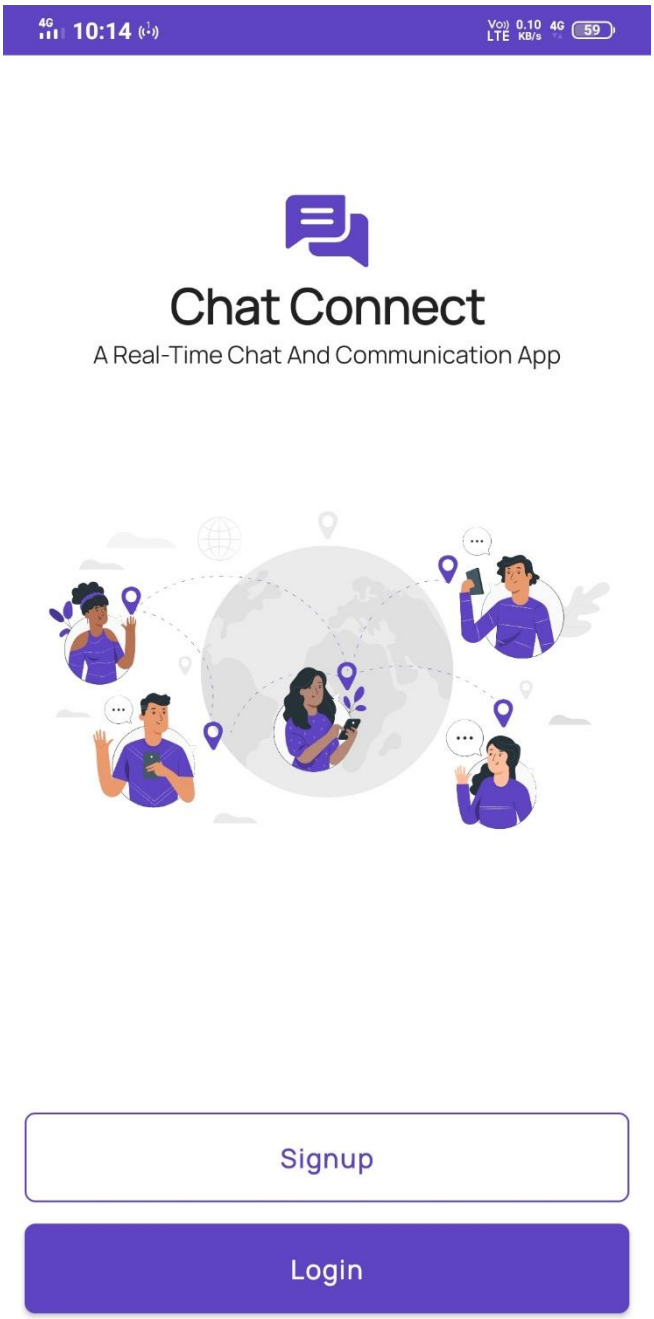
- Select Contact
- Data models
- Language
- Sending Documents
- Location Sharing
- Save States



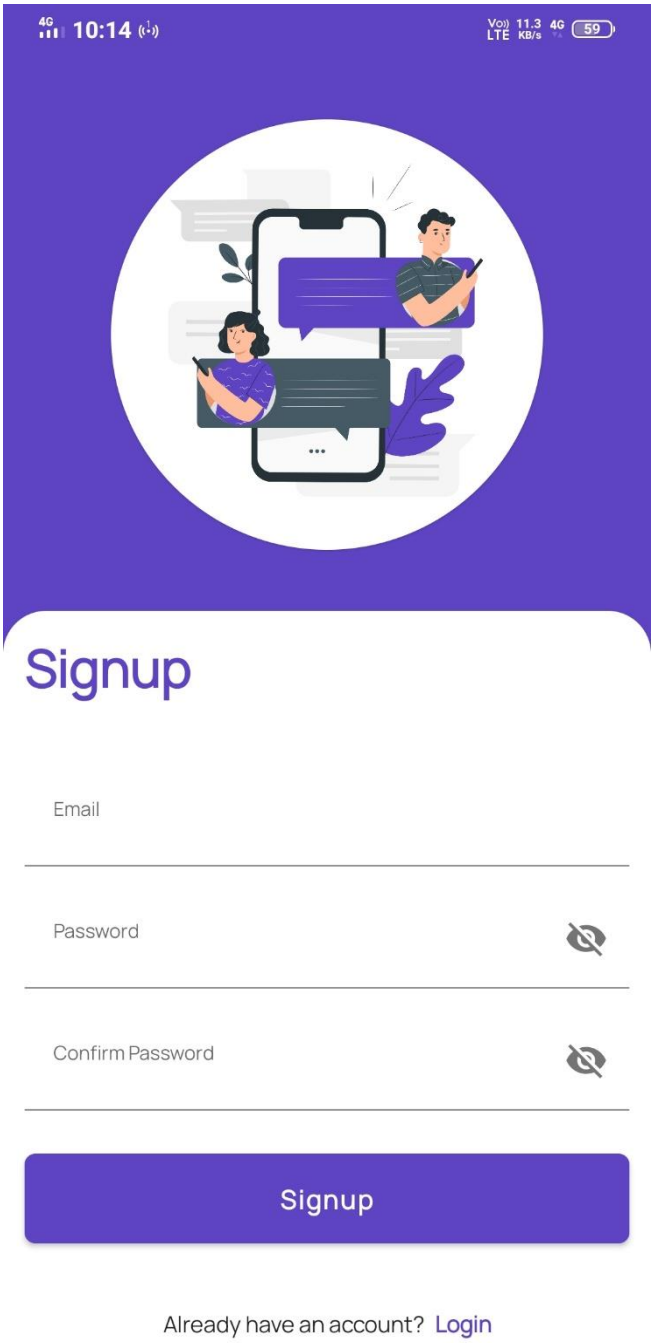
Brainstorming Map https://github.com/karthikeyan9952/chatconnect/documentations

RESULT

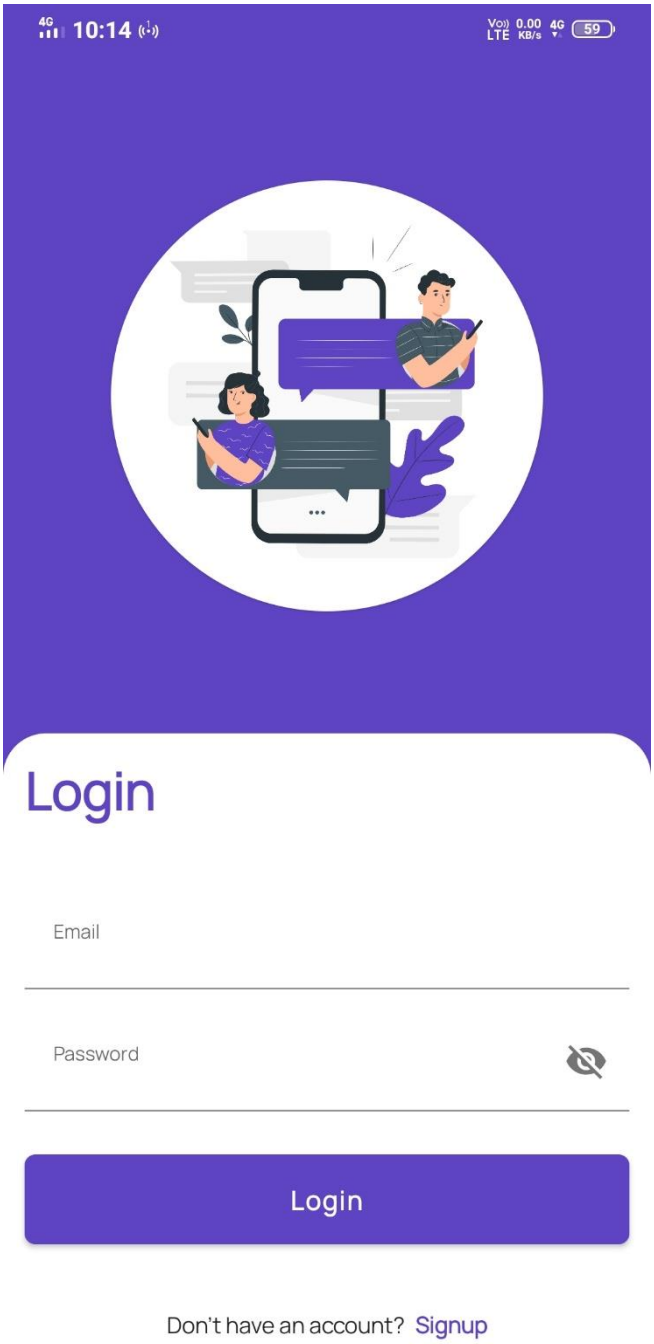
3.1 Screenshots



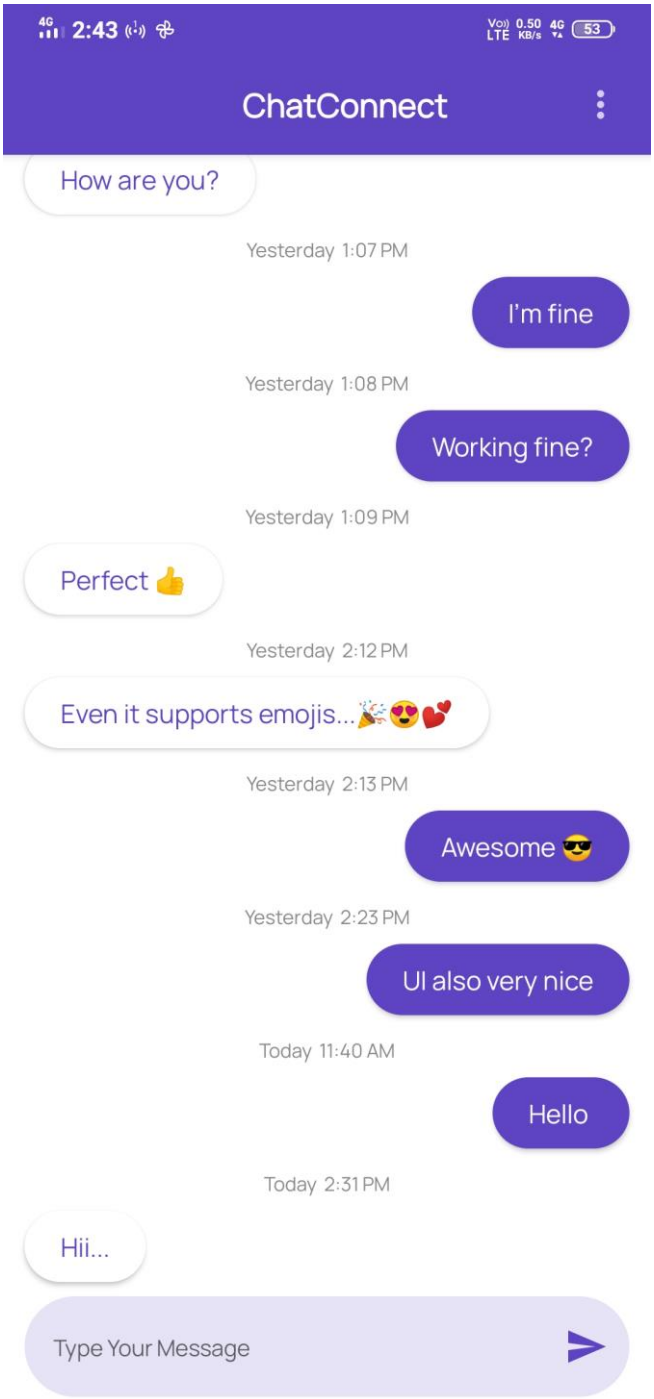
Onboarding Screen



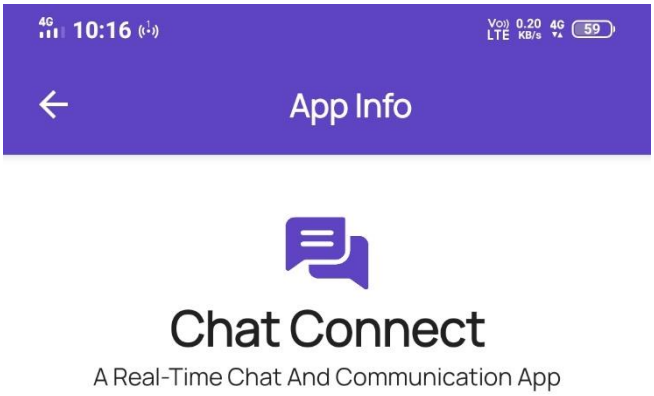
Signup Screen



Login Screen


2:43
ChatConnect
How are you?
Yesterday 1:07 PM
I'm fine
Yesterday 1:08 PM
Working fine?
Yesterday 1:09 PM
Perfect 👍
Yesterday 2:12 PM
Even it supports emojis...🎉😍💕
Yesterday 2:13 PM
Awesome 😎
Yesterday 2:23 PM
UI also very nice
Today 11:40 AM
Hello
Today 2:31 PM
Hii...
Type Your Message

Chat Screen



App Info Screen

ADVANTAGES & DISADVANTAGES

4.1 Advantages

- Ease of communication: ChatConnect allows users to communicate with their contacts in real-time, making it easier for them to stay connected with friends, family, or colleagues.
- User-friendly interface: The user-friendly interface of ChatConnect makes it easy for users to navigate the app and find the features they need.
- Secure communication: The app uses Firebase backend for user authentication, which provides a secure way for users to log in and protect their account information.
- Personalization: The user profile feature in ChatConnect allows users to personalize their profile picture, status, and other personal information to make the app more personalized and engaging.
- Real-time messaging: With Firebase Realtime Database, ChatConnect provides real-time messaging, ensuring that users receive messages as soon as they are sent, which can enhance communication and collaboration.
- Accessibility: ChatConnect is an Android-native app, which means it is accessible to a wide range of users who own an Android device.
- Customization: As the developer, you can customize ChatConnect to suit your user's needs and preferences, adding new features or functionality as needed.

4.2 Disadvantages

- Ease Limited functionality: ChatConnect has limited functionality and is only designed for basic communication and messaging features. Users may prefer to use other apps for more advanced features.
- Network connectivity: ChatConnect requires a stable internet connection to function correctly. Poor network connectivity can result in slow message delivery or lost messages.
- Dependency on Firebase: ChatConnect's backend is dependent on Firebase. If Firebase experiences downtime or other issues, the app may not function correctly.
- Limited platform support: ChatConnect is an Android-native app, which means it is not available on other platforms like iOS or Windows, potentially limiting the user base.
- Security risks: As with any messaging app, there is a risk of users sharing sensitive or personal information, leading to security risks. It is essential to have measures in place to protect user data and privacy.
- User adoption: With so many messaging apps available, it may be challenging to attract and retain users to use ChatConnect as their primary messaging app.
- Maintenance and updates: As the developer, you will need to ensure that the app is maintained and updated regularly to fix bugs, add new features and stay current with platform changes.

APPLICATIONS

- Personal communication: ChatConnect can be used by individuals for personal communication with friends and family, allowing them to stay in touch in real-time.
- Business communication: ChatConnect can be used by businesses for internal communication, allowing employees to collaborate and share information in real-time.
- Education: ChatConnect can be used in educational settings for teacher-student communication, enabling teachers to provide students with timely feedback and support.
- Customer support: ChatConnect can be used by businesses to provide customer support, allowing customers to communicate with support representatives in real-time.
- Social networking: ChatConnect can be used as a social networking platform, enabling users to connect with new people and make friends.
- Group communication: ChatConnect can be used for group communication, allowing users to create groups for different interests and topics, and communicate with multiple people at once.
- Community-building: ChatConnect can be used to build online communities around specific interests, hobbies, or causes, enabling users to connect with like-minded people and share ideas and experiences.

CONCLUSION

ChatConnect is a chatting application developed for Android using Kotlin and Firebase backend. It is designed to enable real-time communication between users, allowing them to chat and exchange messages in a secure and user-friendly manner. ChatConnect provides features such as user authentication, real-time messaging, user profile, and a simple messaging interface.

While ChatConnect has several potential advantages, such as ease of communication, personalization, and real-time messaging, it also has a few potential disadvantages, including limited functionality, network connectivity dependency, and limited platform support.

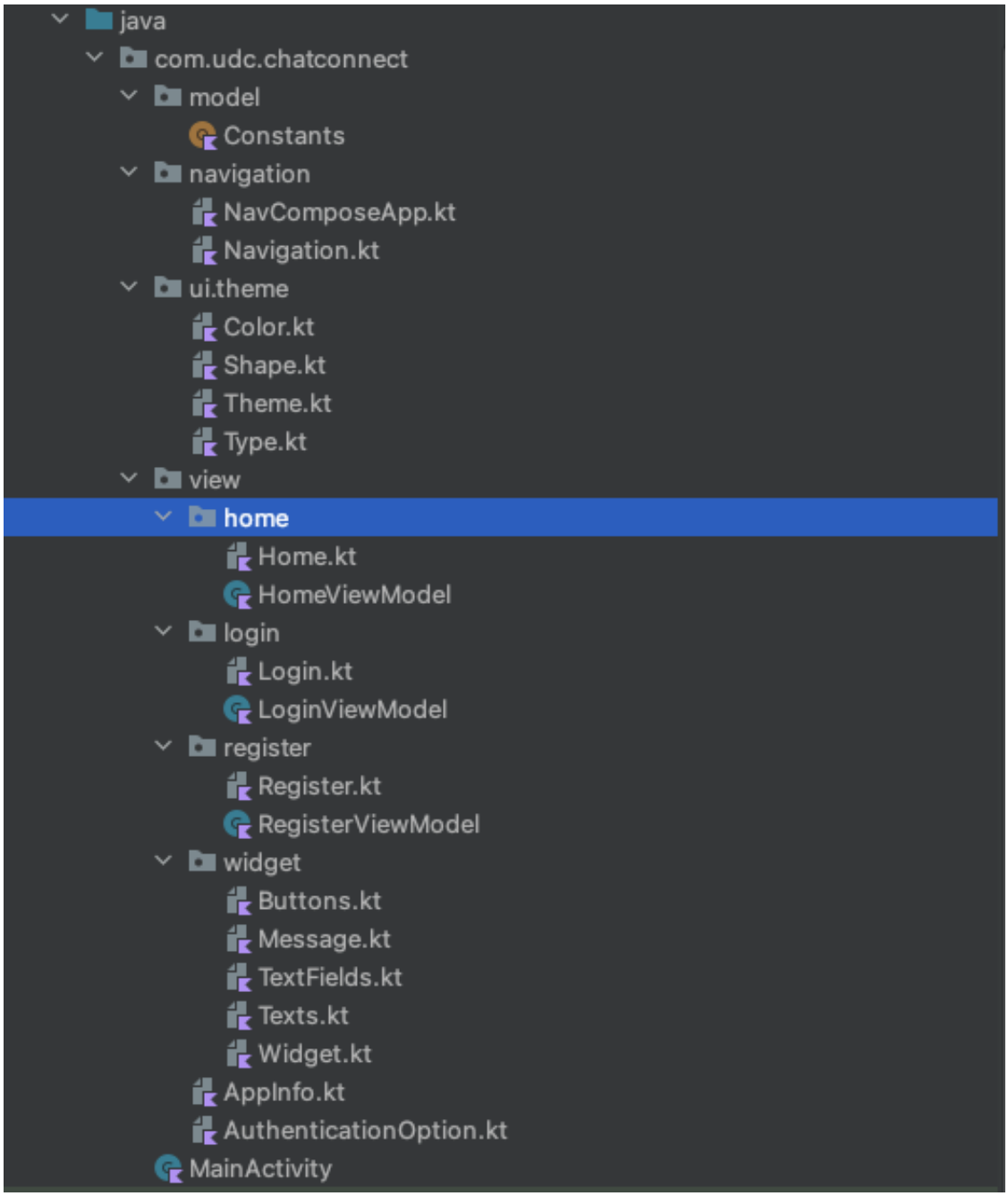
Overall, ChatConnect can be used for a variety of applications, including personal communication, business communication, education, customer support, social networking, group communication, and community-building. With ongoing maintenance and updates, ChatConnect has the potential to be a useful tool for facilitating communication and collaboration among its users.

FUTURE SCOPE

1. Location sharing: By adding location sharing features, users can share their location with friends or family members, making it easier to meet up or find each other in real-time.
2. File sharing: The ability to share files through ChatConnect can be useful for businesses or individuals who need to share documents or images quickly and easily.
3. Group messaging: Group messaging allows users to communicate with multiple people at once, which can be useful for collaboration, event planning, or socializing.
4. Voice and video calling: By adding voice and video calling features, ChatConnect can compete with other popular messaging apps and offer users more ways to communicate with their contacts.
5. Advanced security features: With the increasing concerns over online privacy and security, adding advanced security features such as end-to-end encryption, two-factor authentication, or self-destructing messages can attract more users and enhance the app's reputation.
6. Artificial Intelligence: Implementing AI in the ChatConnect app can add more personalization features such as intelligent chatbots, predictive text, or personalized recommendations.

By adding these features, ChatConnect can become a more comprehensive and competitive messaging app, attracting more users and increasing user engagement.

APPENDIX



Folder Structure

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.Chatconnect"
        tools:targetApi="31">

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.Chatconnect">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

build.gradle - app

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'com.google.gms.google-services'
}

android {
    namespace 'com.udc.chatconnect'
    compileSdk 33

    defaultConfig {
        applicationId "com.udc.chatconnect"
        minSdk 24
        targetSdk 33
        versionCode 1
        versionName '1.1'

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
```

```
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion '1.2.0'
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {
    implementation 'androidx.compose.ui:ui-text-google-fonts:1.2.1'
    implementation "org.jetbrains.compose.material:material-icons-extended-desktop:$compose_ui_version"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.material:material:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
    implementation "androidx.compose.runtime:runtime-livedata:$compose_ui_version"
    implementation 'androidx.lifecycle:lifecycle-viewmodel-compose:1.0.0-alpha07'
    implementation "androidx.navigation:navigation-compose:2.4.0-alpha06"
    implementation "com.google.accompanist:accompanist-systemuicontroller:0.27.0"

    implementation platform('com.google.firebase:firebase-bom:28.3.0')
    implementation 'com.google.firebase:firebase-analytics-ktx'
    implementation 'com.google.firebase:firebase-auth-ktx'
    implementation 'com.google.firebase:firebase-firestore-ktx'
    implementation 'com.google.firebase:firebase-auth:21.0.3'
    implementation 'com.google.firebase:firebase-firestore:24.1.1'

    testImplementation 'junit:junit:4.13.2'
```

```
    androidTestImplementation                                   'androidx.test.ext:junit:1.1.3'
    androidTestImplementation                                   'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation                                   "androidx.compose.ui:ui-test-junit4:$compose_ui_version"

    debugImplementation                                         "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation                                         "androidx.compose.ui:ui-test-manifest:$compose_ui_version"
}
```

build.gradle - project

```
buildscript {
    ext {
        compose_ui_version = '1.2.0'
    }
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:7.0.0"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.21"
        classpath "com.google.gms:google-services:4.3.8"
//        classpath 'com.google.gms:google-services:4.3.15'
    }
}// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.4.2' apply false
    id 'com.android.library' version '7.4.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.0' apply false
}
```

MainActivity.kt

```
package com.udc.chatconnect

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.FirebaseApp
import com.udc.chatconnect.navigation.NavComposeApp
import com.udc.chatconnect.view.widget.toastMessage

class MainActivity : ComponentActivity() {
    private var backPressedTime = 0L
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        FirebaseApp.initializeApp(this)
        setContent {
            NavComposeApp()
        }
    }

    override fun onBackPressed() {
        if (backPressedTime + 2000 > System.currentTimeMillis()){
            super.onBackPressed()
        }else{
            toastMessage("Tap again to exit app", context = applicationContext)
        }
        backPressedTime = System.currentTimeMillis()
    }
}
```

NavComposeApp.kt

```
package com.udc.chatconnect.navigation

import AuthenticationView
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.udc.chatconnect.navigation.Destination.AppInfo
import com.udc.chatconnect.navigation.Destination.AuthenticationOption
import com.udc.chatconnect.navigation.Destination.Home
import com.udc.chatconnect.navigation.Destination.Login
import com.udc.chatconnect.navigation.Destination.Register
import com.udc.chatconnect.ui.theme.ChatconnectTheme
import com.udc.chatconnect.view.AppInfoView
import com.udc.chatconnect.view.home.HomeView
import com.udc.chatconnect.view.login.LoginView
import com.udc.chatconnect.view.register.RegisterView

@Composable
fun NavComposeApp() {
    val navController = rememberNavController()
    val actions = remember(navController) { Action(navController) }
    ChatconnectTheme {
        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.forwardHomeInRegister,
                    login = actions.replaceLoginWithRegister
                )
            }
            composable(Login) {
                LoginView(
                    home = actions.forwardHomeInLogin,
                    register = actions.replaceRegisterWithLogin,
                    pop = actions.navigateBack
                )
```

```
        }
        composable(Home) {
            HomeView(landing = actions.gotoLanding, appInfo = actions.goToAppInfo)
        }
        composable(AppInfo) {
            AppInfoView(back = actions.navigateBack)
        }
    }
  }
}
```

Navigation.kt

```
package com.udc.chatconnect.navigation

import androidx.navigation.NavHostController
import androidx.navigation.NavOptions
import com.udc.chatconnect.navigation.Destination.AppInfo
import com.udc.chatconnect.navigation.Destination.AuthenticationOption
import com.udc.chatconnect.navigation.Destination.Home
import com.udc.chatconnect.navigation.Destination.Login
import com.udc.chatconnect.navigation.Destination.Register


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

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() }

    val replaceLoginWithRegister: () -> Unit = {  navController.navigate(
```

```
        Login,
        NavOptions.Builder()
            .setPopUpTo(Register, inclusive = true)
            .setLaunchSingleTop(true)
            .build()
    ) }

    val replaceRegisterWithLogin: () -> Unit = {  navController.navigate(
        Register,
        NavOptions.Builder()
            .setPopUpTo(Login, inclusive = true)
            .setLaunchSingleTop(true)
            .build()
    ) }

    val gotoLanding: () -> Unit = {  navController.navigate(
        AuthenticationOption,
        NavOptions.Builder()
            .setPopUpTo(Home, inclusive = true)
            .setLaunchSingleTop(true)
            .build()
    ) }

    val forwardHomeInLogin: () -> Unit = {  navController.navigate(
        Home,
        NavOptions.Builder()
            .setPopUpTo(Login, inclusive = true)
            .setLaunchSingleTop(true)
            .build()
    ) }

    val forwardHomeInRegister: () -> Unit = {  navController.navigate(
        Home,
        NavOptions.Builder()
            .setPopUpTo(Register, inclusive = true)
            .setLaunchSingleTop(true)
            .build()
    ) }

    val goToAppInfo: () -> Unit = { navController.navigate(AppInfo) }
}
```

Constants.kt

```
package com.udc.chatconnect.model

object Constants {
    const val TAG = "chat-connect"

    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"
}
```

Color.kt

```
package com.udc.chatconnect.ui.theme

import androidx.compose.ui.graphics.Color

val Primary = Color(0xFF5e43c3)
val Purple200 = Color(0xFFBB86FC)
val Purple500 = Color(0xFF6200EE)
val Purple700 = Color(0xFF3700B3)
val Teal200 = Color(0xFF03DAC5)
val Dark = Color(0xFF212121)
```

Shape.kt

```
package com.udc.chatconnect.ui.theme

import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Shapes
import androidx.compose.ui.unit.dp

val Shapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(4.dp),
    large = RoundedCornerShape(0.dp)
)
```

Theme.kt

```
package com.udc.chatconnect.ui.theme

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material.MaterialTheme
import androidx.compose.material.darkColors
import androidx.compose.material.lightColors
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.Color
import com.google.accompanist.systemuicontroller.rememberSystemUiController

private val DarkColorPalette = darkColors(
    primary = Primary,
    primaryVariant = Purple700,
    secondary = Teal200,
    background = Color.White
)

private val LightColorPalette = lightColors(
    primary = Primary,
    primaryVariant = Purple700,
    secondary = Teal200,
    background = Color.White

    /* Other default colors to override
    background = Color.White,
    surface = Color.White,
    onPrimary = Color.White,
    onSecondary = Color.Black,
    onBackground = Color.Black,
    onSurface = Color.Black,
    */
)

@Composable
fun ChatconnectTheme(darkTheme: Boolean = isSystemInDarkTheme(), content: @Composable () -> Unit) {
//    val colors = if (darkTheme) {
//        DarkColorPalette
//    } else {
//        LightColorPalette
//    }
    val colors = LightColorPalette

    val systemUiController = rememberSystemUiController()
    if(darkTheme){
        systemUiController.setSystemBarsColor(
            color = Primary
        )
    }else{
        systemUiController.setSystemBarsColor(
            color = Primary
        )
    }
```

```
    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

Type.kt

```
package com.udc.chatconnect.ui.theme

import androidx.compose.material.Typography
import androidx.compose.ui.text.ExperimentalTextApi
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.googlefonts.Font
import androidx.compose.ui.text.googlefonts.GoogleFont
import androidx.compose.ui.unit.sp
import com.udc.chatconnect.R

// Set of Material typography styles to start with
val Typography = Typography(
    body1 = TextStyle(
        fontFamily = FontFamily.SansSerif,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
    )
    /* Other default text styles to override
    button = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.W500,
        fontSize = 14.sp
    ),
    caption = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 12.sp
    )
    */
)

@OptIn(ExperimentalTextApi::class)
val provider = GoogleFont.Provider(
    providerAuthority = "com.google.android.gms.fonts",
    providerPackage = "com.google.android.gms",
    certificates = R.array.com_google_android_gms_fonts_certs
)
```

```
// GoogleFont.Provider initialization ...

@OptIn(ExperimentalTextApi::class)
val pacificoFontName = GoogleFont("Pacifico")
@OptIn(ExperimentalTextApi::class)
val manropeFontName = GoogleFont("Manrope")


@OptIn(ExperimentalTextApi::class)
val pacifico = FontFamily(Font(googleFont = pacificoFontName, fontProvider = provider))
@OptIn(ExperimentalTextApi::class)
val manrope = FontFamily(Font(googleFont = manropeFontName, fontProvider = provider))
```

Home.kt

```
package com.udc.chatconnect.view.home

import android.annotation.SuppressLint
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.udc.chatconnect.model.Constants
import com.udc.chatconnect.view.widget.Appbar
import com.udc.chatconnect.view.widget.SingleMessageWithDate
import com.udc.chatconnect.view.widget.TextFieldMessage
import java.text.SimpleDateFormat
import java.util.*


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun HomeView(
    homeViewModel: HomeViewModel = viewModel(),
    landing: () -> Unit,
    appInfo: () -> Unit
) {
    val message: String by homeViewModel.message.observeAsState(initial = "")
    val messages: List<Map<String, Any>> by homeViewModel.messages.observeAsState(
        initial = emptyList<Map<String, Any>>().toMutableList()
    )
    val context = LocalContext.current
    var tmpDate = SimpleDateFormat("MM/dd/yyyy").parse("10/10/2023")
```

```
    Scaffold(
        topBar = {
            Appbar(
                title = "ChatConnect",
                logout = { homeViewModel.logoutUser(landing = landing, context = context) },
                about = appInfo
            )
        },
        bottomBar = {
            TextFieldMessage(
                message = message,
                onValueChange = { homeViewModel.updateMessage(it) },
                send = { homeViewModel.addMessage() })
        }
    ) {
        LazyColumn(
            modifier = Modifier
                .fillMaxSize()
                .padding(bottom = 60.dp),
            contentPadding = PaddingValues(horizontal = 12.dp, vertical = 8.dp),
            verticalArrangement = Arrangement.spacedBy(4.dp),
            reverseLayout = true
        ) {
            itemsIndexed(messages) { index, message ->
                val isCurrentUser = message[Constants.IS_CURRENT_USER] as Boolean
                val timeStamp = message[Constants.SENT_ON].toString().toLong()

                SingleMessageWithDate(
                    message = message[Constants.MESSAGE].toString(),
                    timestamp = homeViewModel.convertLongToTime(timeStamp),
                    isCurrentUser = isCurrentUser,
                    isLast = index == 0,
                    date = homeViewModel.convertLongToTime(timeStamp)
                )

            }
        }
    }
}
```

HomeViewModel.kt

```
package com.udc.chatconnect.view.home


import android.content.Context
import android.icu.text.SimpleDateFormat
import android.util.Log
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.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import com.udc.chatconnect.model.Constants
import com.udc.chatconnect.view.widget.toastMessage
import java.lang.IllegalArgumentException
import java.util.*

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

    private val _isSending = MutableLiveData(false)
    val isSending: LiveData<Boolean> = _isSending

    private val auth: FirebaseAuth = Firebase.auth

    /**
     * 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()) {
            _isSending.value = true
            _message.value = ""
```

```
        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 {
            _isSending.value = false
        }
    }
}

/**
 * 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 logoutUser(landing: () -> Unit, context: Context) {
    auth.signOut()
    landing()
    toastMessage("Logout Successful", context)
}

fun convertLongToTime(time: Long): String {
```

```
        val today = Date()
        val date = Date(time)
        var hour: Int = date.hours
        var minutes: Int = date.minutes
        var suffix: String = ""

        if (hour > 11) {
            suffix = "PM"
            if (hour > 12)
                hour -= 12
        } else {
            suffix = "AM"
            if (hour == 0)
                hour = 12
        }

        val t = if (minutes < 10) "$hour:0$minutes $suffix" else "$hour:$minutes $suffix"

        val sdf = SimpleDateFormat("dd-MM-yyyy")
        val formattedDate = sdf.format(date)

        val diff: Long = today.time - date.time
        val seconds = diff / 1000
        val minute = seconds / 60
        val hours = minute / 60
        val days = hours / 24

        return if (days == 0L) "Today  $t" else if (days == 1L) "Yesterday  $t" else "$formattedDate  $t"

    }

}
```

Login.kt

```
package com.udc.chatconnect.view.login

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Card
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.platform.LocalContext
import androidx.compose.ui.res.painterResource
```

```
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.udc.chatconnect.R
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.view.widget.Loader
import com.udc.chatconnect.view.widget.toastMessage
import com.udc.chatconnect.view.widget.*



@Composable
fun LoginView(
    home: () -> Unit,
    pop: () -> 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)
    val isVisible: Boolean by loginViewModel.isVisible.observeAsState(false)
    val context = LocalContext.current



    Column(
        verticalArrangement = Arrangement.SpaceBetween,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .background(color = Primary)

    ) {
        Spacer(modifier = Modifier.size(height = 2.dp, width = 0.dp))
        Card(
            modifier = Modifier.size(240.dp),
            shape = RoundedCornerShape(300.dp),

            ) {
            Image(
                modifier = Modifier.padding(8.dp),
                painter = painterResource(R.drawable.messages), contentDescription = null
            )
        }

        Card(
            shape = RoundedCornerShape(topEnd = 24.dp, topStart = 24.dp),
        ) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(12.dp)
            ) {
                TextH1(value = "Login")
                Spacer(modifier = Modifier.size(height = 36.dp, width = 0.dp))
                TextFieldAuth(
                    text = email,
```

```
            onValueChange = { loginViewModel.updateEmail(it) },
            label = "Email"
        )
        Spacer(modifier = Modifier.size(height = 12.dp, width = 0.dp))
        TextFieldPassword(
            text = password,
            onValueChange = { loginViewModel.updatePassword(it) },
            toggleVisible = { loginViewModel.toggleIsVisible() },
            label = "Password",
            isVisible = isVisible
        )
        Spacer(modifier = Modifier.size(height = 24.dp, width = 0.dp))
        if (loading) Loader() else ButtonPrimary(
            title = "Login",
            onClick = {
                if (email.isEmpty()) toastMessage(
                    "Email is empty",
                    context
                ) else if (password.isEmpty()) toastMessage(
                    "Password is empty",
                    context
                ) else loginViewModel.loginUser(home = home)
            })
        Spacer(modifier = Modifier.size(height = 36.dp, width = 0.dp))
        Row(
            modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.Center
        ) {
            PText(text = "Don't have an account?")
            Spacer(modifier = Modifier.size(height = 0.dp, width = 6.dp))
            ClickableTxt(text = "Signup", onClick = register)
        }
        Spacer(modifier = Modifier.size(height = 24.dp, width = 0.dp))
    }
  }

 }

}
```

LoginViewModel.kt

```
package com.udc.chatconnect.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

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

    private val _isVisible = MutableLiveData(false)
    val isVisible: LiveData<Boolean> = _isVisible

    // Update email
    fun updateEmail(newEmail: String) {
        _email.value = newEmail
    }

    // Update password
    fun updatePassword(newPassword: String) {
        _password.value = newPassword
    }

    fun toggleIsVisible() {
        _isVisible.value = !_isVisible.value!!
    }

    // Register user
    fun loginUser(home: () -> Unit) {
        if (_loading.value == false) {
            val email: String = _email.value ?: throw IllegalArgumentException("email expected")
            val password: String =
                _password.value ?: throw IllegalArgumentException("password expected")

            _loading.value = true

            auth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener {
                    if (it.isSuccessful) {
                        home()
                    }
                    _loading.value = false
                }
        }
    }
}
```

Register.kt

```
package com.udc.chatconnect.view.register

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Card
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.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.udc.chatconnect.R
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.view.widget.*



@Composable
fun RegisterView(
    home: () -> Unit,
    login: () -> Unit,
    registerViewModel: RegisterViewModel = viewModel()
) {
    val email: String by registerViewModel.email.observeAsState("")
    val password: String by registerViewModel.password.observeAsState("")
    val confirmpassword: String by registerViewModel.confirmpassword.observeAsState("")
    val isVisible: Boolean by registerViewModel.isVisible.observeAsState(false)
    val isVisibleConfirm: Boolean by registerViewModel.isVisibleConfrim.observeAsState(false)
    val loading: Boolean by registerViewModel.loading.observeAsState(initial = false)
    val context = LocalContext.current

    Column(
        verticalArrangement = Arrangement.SpaceBetween,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .background(color = Primary)

    ) {
        Spacer(modifier = Modifier.size(height = 2.dp, width = 0.dp))
        Card(
            modifier = Modifier.size(240.dp),
            shape = RoundedCornerShape(300.dp),

            ) {
            Image(
                modifier = Modifier.padding(8.dp),
```

```
        painter = painterResource(R.drawable.messages), contentDescription = null
    )
}

Card(
    shape = RoundedCornerShape(topEnd = 24.dp, topStart = 24.dp),
    backgroundColor = Color.White
) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(12.dp)
    ) {
        TextH1(value = "Signup")
        Spacer(modifier = Modifier.size(height = 36.dp, width = 0.dp))
        TextFieldAuth(
            text = email,
            onValueChange = { registerViewModel.updateEmail(it) },
            label = "Email"
        )
        Spacer(modifier = Modifier.size(height = 12.dp, width = 0.dp))
        TextFieldPassword(
            text = password,
            onValueChange = { registerViewModel.updatePassword(it) },
            toggleVisible = { registerViewModel.toggleIsVisible() },
            label = "Password",
            isVisible = isVisible
        )
        Spacer(modifier = Modifier.size(height = 12.dp, width = 0.dp))
        TextFieldPassword(
            text = confirmpassword,
            onValueChange = { registerViewModel.updateConfirmPassword(it) },
            toggleVisible = { registerViewModel.toggleIsVisibleConfirmation() },
            label = "Confirm Password",
            isVisible = isVisibleConfirm
        )
        Spacer(modifier = Modifier.size(height = 24.dp, width = 0.dp))
        if (loading) Loader() else ButtonPrimary(
            title = "Signup",
            onClick = {
                if (email.isEmpty()) toastMessage(
                    "Email is Empty",
                    context
                ) else if (password.isEmpty()) toastMessage(
                    "Password is empty",
                    context
                ) else if (password != confirmpassword) toastMessage(
                    "Passwords not matching",
                    context
                ) else registerViewModel.registerUser(home = home)
            })
        Spacer(modifier = Modifier.size(height = 36.dp, width = 0.dp))
        Row(
            modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.Center
        ) {
            PText(text = "Already have an account?")
```

```
            Spacer(modifier = Modifier.size(height = 0.dp, width = 6.dp))
            ClickableTxt(text = "Login", onClick = login)
        }
        Spacer(modifier = Modifier.size(height = 24.dp, width = 0.dp))
    }
  }

 }

}
```

RegisterViewModel.kt

```
package com.udc.chatconnect.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

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 _confirmpassword = MutableLiveData("")
    val confirmpassword: LiveData<String> = _confirmpassword

    private val _loading = MutableLiveData(false)
    val loading: LiveData<Boolean> = _loading

    private val _isVisible = MutableLiveData(false)
    val isVisible: LiveData<Boolean> = _isVisible

    private val _isVisibleConfirm = MutableLiveData(false)
    val isVisibleConfrim: LiveData<Boolean> = _isVisibleConfirm

    // Update email
    fun updateEmail(newEmail: String) {
        _email.value = newEmail
    }
```

```
    // Update password
    fun updatePassword(newPassword: String) {
        _password.value = newPassword
    }

    fun updateConfirmPassword(newPassword: String) {
        _confirmpassword.value = newPassword
    }

    fun toggleIsVisible() {
        _isVisible.value = !_isVisible.value!!
    }

    fun toggleIsVisibleConfirmation() {
        _isVisibleConfirm.value = !_isVisibleConfirm.value!!
    }

    // 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
                }
        }
    }
}
```

Buttons.kt

```
package com.udc.chatconnect.view.widget

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Button
import androidx.compose.material.ButtonDefaults
import androidx.compose.material.OutlinedButton
import androidx.compose.material.Text
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.unit.dp
import androidx.compose.ui.unit.sp
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope


@Composable
fun ButtonPrimary(title: String, onClick: () -> Unit) {
    Button(
        onClick = onClick,
        colors = ButtonDefaults.buttonColors(
            backgroundColor = Primary, contentColor = Color.White
        ),
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp),
        shape = RoundedCornerShape(12),
    ) {
        Text(
            text = title, fontFamily = manrope, fontSize = 16.sp, fontWeight = FontWeight.SemiBold
        )
    }
}

@Composable
fun ButtonSecondary(title: String, onClick: () -> Unit) {
    OutlinedButton(
        onClick = onClick,
        border = BorderStroke(1.dp, Primary),
        colors = ButtonDefaults.buttonColors(
            backgroundColor = Color.Transparent, contentColor = Color.White
        ),
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp),
        shape = RoundedCornerShape(12),
```

```
    ) {
        Text(
            text = title,
            fontFamily = manrope,
            fontSize = 16.sp,
            color = Primary,
            fontWeight = FontWeight.SemiBold
        )
    }
}
```

Message.kt

```
package com.udc.chatconnect.view.widget

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Card
import androidx.compose.material.Icon
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.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.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope
import com.udc.chatconnect.view.home.HomeViewModel


@Composable
fun SingleMessage(
    homeViewModel: HomeViewModel = viewModel(),
    message: String,
    timestamp: String,
    isCurrentUser: Boolean,
    isLast: Boolean
) {
    val isSending: Boolean by homeViewModel.isSending.observeAsState(false)
```

```
    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = if (isCurrentUser) Arrangement.End else Arrangement.Start,
        verticalAlignment = Alignment.Bottom
    ) {
        Card(
            shape = if (isCurrentUser) RoundedCornerShape(
                topStart = 32.dp,
                topEnd = 0.dp,
                bottomStart = 32.dp,
                bottomEnd = 32.dp
            ) else RoundedCornerShape(
                topStart = 0.dp,
                topEnd = 32.dp,
                bottomStart = 32.dp,
                bottomEnd = 32.dp
            ),
            backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else Color.White,
            elevation = 2.dp
        ) {
            Row(
                modifier = Modifier
                    .padding(vertical = 14.dp, horizontal = 18.dp),
                verticalAlignment = Alignment.Bottom
            ) {
                Text(
                    text = message,
                    fontFamily = manrope,
                    textAlign = if (isCurrentUser) TextAlign.End
                    else TextAlign.Start,
                    color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White
                )
                Spacer(modifier = Modifier.width(8.dp))
                Text(
                    text = timestamp,
                    fontFamily = manrope,
                    textAlign = if (isCurrentUser) TextAlign.End
                    else TextAlign.Start,
                    color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White,
                    fontSize = 10.sp
                )
            }
        }
        if (isCurrentUser && isLast && isSending) Icon(
            Icons.Outlined.Send,
            contentDescription = "Sending",
            tint = Primary
        ) else Spacer(
            modifier = Modifier.width(0.dp)
        )
    }
}

@Composable
fun SingleMessageWithDate(
    homeViewModel: HomeViewModel = viewModel(),
```

```
    message: String,
    timestamp: String,
    isCurrentUser: Boolean,
    isLast: Boolean,
    date: String
) {
    val isSending: Boolean by homeViewModel.isSending.observeAsState(false)

    Column {
        Spacer(modifier = Modifier.height(8.dp))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.Center
        ) {
            Text(
                text = date,
                fontFamily = manrope,
                textAlign = TextAlign.Center,
                fontSize = 12.sp,
                color = Color.Gray
            )
        }
        Spacer(modifier = Modifier.height(8.dp))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = if (isCurrentUser) Arrangement.End else Arrangement.Start,
            verticalAlignment = Alignment.Bottom
        ) {
            Card(
                shape = if (isCurrentUser) RoundedCornerShape(
                    topStart = 24.dp,
                    topEnd = 24.dp,
                    bottomStart = 24.dp,
                    bottomEnd = 24.dp
                ) else RoundedCornerShape(
                    topStart = 24.dp,
                    topEnd = 24.dp,
                    bottomStart = 24.dp,
                    bottomEnd = 24.dp
                ),
                backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else Color.White,
                elevation = 2.dp
            ) {
                Row(
                    modifier = Modifier
                        .padding(vertical = 10.dp, horizontal = 20.dp),
                    verticalAlignment = Alignment.Bottom
                ) {
                    Text(
                        text = message,
                        fontFamily = manrope,
                        textAlign = if (isCurrentUser) TextAlign.End
                        else TextAlign.Start,
                        color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White
                    )
                }
```

```
            }
            if (isCurrentUser && isLast && isSending) Icon(
                Icons.Outlined.Send,
                contentDescription = "Sending",
                tint = Primary
            ) else Spacer(
                modifier = Modifier.width(0.dp)
            )
        }
    }
}
```

TextFields.kt

```
package com.udc.chatconnect.view.widget

import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardActions
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.material.icons.filled.Visibility
import androidx.compose.material.icons.filled.VisibilityOff
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.focus.FocusDirection
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.text.input.*
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope

@Composable
fun TextFieldMessage(message: String, onValueChange: (String) -> Unit, send: () -> Unit) {
    TextField(
        value = message,
        onValueChange = onValueChange,
        placeholder = {
            Text("Type Your Message", fontFamily = manrope, fontSize = 14.sp)
        },
        maxLines = 1,
        modifier = Modifier
            .padding(horizontal = 12.dp, vertical = 4.dp)
            .fillMaxWidth(),
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Text,
```

```
            imeAction = ImeAction.None,
            capitalization = KeyboardCapitalization.Sentences,
        ),
        singleLine = true,
        trailingIcon = {
            IconButton(
                onClick = send
            ) {
                Icon(
                    imageVector = Icons.Default.Send,
                    contentDescription = "Send Button",
                    tint = Primary
                )
            }
        },
        shape = RoundedCornerShape(32.dp),
        colors = TextFieldDefaults.textFieldColors(
            backgroundColor = Primary.copy(alpha = 0.15f),
            focusedIndicatorColor = Color.Transparent,
            unfocusedIndicatorColor = Color.Transparent,
            disabledIndicatorColor = Color.Transparent,
            errorIndicatorColor = Color.Transparent,
            cursorColor = Primary,
        ),
    )
}

@Composable
fun TextFieldAuth(
    text: String,
    onValueChange: (String) -> Unit,
    label: String
) {
    val focusManager = LocalFocusManager.current
    TextField(
        modifier = Modifier.fillMaxWidth(),
        value = text,
        colors = TextFieldDefaults.textFieldColors(
            backgroundColor = MaterialTheme.colors.background,
        ),
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Email,
            imeAction = ImeAction.Done
        ),
        keyboardActions = KeyboardActions(onDone = {focusManager.clearFocus()}),
        label = {
            LabelTxt(text = label)
        },
        onValueChange = onValueChange
    )
}


@Composable
fun TextFieldPassword(
    text: String,
```

```
    onValueChange: (String) -> Unit,
    toggleVisible: () -> Unit,
    label: String,
    isVisible: Boolean
) {
    val focusManager = LocalFocusManager.current
    TextField(
        modifier = Modifier.fillMaxWidth(),
        value = text,
        colors = TextFieldDefaults.textFieldColors(
            backgroundColor = MaterialTheme.colors.background,
        ),
        label = { LabelTxt(text = label) },
        visualTransformation = if (isVisible) VisualTransformation.None else PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Password,
            imeAction = ImeAction.Done
        ),
        keyboardActions = KeyboardActions(onDone = {focusManager.clearFocus()}),
        trailingIcon = {
            IconButton(onClick = toggleVisible) {
                Icon(
                    imageVector = if (!isVisible) Icons.Filled.VisibilityOff else Icons.Filled.Visibility,
                    contentDescription = ""
                )
            }
        },
        onValueChange = onValueChange
    )
}
```

Texts.kt

```
package com.udc.chatconnect.view.widget

import androidx.compose.foundation.clickable
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope

@Composable
fun Title(title: String) {
    Text(
```

```
        text = title, fontSize = 30.sp, fontWeight = FontWeight.Bold, fontFamily = manrope
    )
}

@Composable
fun Description(desc: String) {
    Text(
        text = desc, fontSize = 15.sp, fontFamily = manrope
    )
}

@Composable
fun ClickableTxt(text: String, onClick: () -> Unit) {
    Text(
        modifier = Modifier.clickable(onClick = onClick),
        text = text,
        fontSize = 14.sp,
        color = Primary,
        fontWeight = FontWeight.SemiBold,
        fontFamily = manrope
    )
}

@Composable
fun PText(text: String) {
    Text(
        text = text, fontSize = 14.sp, fontFamily = manrope
    )
}

@Composable
fun LabelTxt(text: String) {
    Text(text = text, fontSize = 12.sp, fontFamily = manrope)
}

@Composable
fun TextH1(value: String) {
    Text(
        text = value,
        fontSize = 32.sp,
        fontWeight = FontWeight.Bold,
        color = Primary,
        fontFamily = manrope
    )
}
```

Widget.kt

```
package com.udc.chatconnect.view.widget


import android.content.Context
import android.widget.Toast
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.material.icons.outlined.Info
import androidx.compose.material.icons.outlined.Logout
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope


@Composable
fun Appbar(title: String, logout: () -> Unit, about: () -> Unit) {
    var showMenu by remember { mutableStateOf(false) }
    TopAppBar(
        title = {
            Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.Center) {
                Text(text = title, fontFamily = manrope, fontWeight = FontWeight.Bold)
            }
        },
        actions = {
            IconButton(onClick = { showMenu = true }) {
                Icon(Icons.Filled.MoreVert, contentDescription = "Menu")
            }
            DropdownMenu(modifier = Modifier.padding(horizontal = 14.dp),
                expanded = showMenu,
                onDismissRequest = { showMenu = false }
            ) {
                DropdownMenuItem(onClick = about) {
                    Row {
                        Text("About", fontFamily = manrope, fontSize = 16.sp)
                        Spacer(modifier = Modifier.width(8.dp))
                        Icon(Icons.Outlined.Info, contentDescription = "Logout", tint = Primary)
                    }
                }
                DropdownMenuItem(onClick = logout) {
                    Row {
                        Text("Logout", fontFamily = manrope, fontSize = 16.sp)
                        Spacer(modifier = Modifier.width(8.dp))
                        Icon(Icons.Outlined.Logout, contentDescription = "Logout", tint = Primary)
                    }
```

```
            }
        }
    },
    navigationIcon = { Spacer(modifier = Modifier.size(24.dp)) }
  )
}


@Composable
fun Loader() {
  Row(
    horizontalArrangement = Arrangement.Center,
    modifier = Modifier.fillMaxWidth()
  ) {
    CircularProgressIndicator()
  }
}


fun toastMessage(text: String, context: Context) {
  Toast.makeText(
    context,
    text,
    Toast.LENGTH_SHORT
  ).show()
}
```

AuthenticationOption.kt

```
package com.udc.chatconnect.view

import android.annotation.SuppressLint
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
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.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.ColorFilter
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.udc.chatconnect.R
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope
import com.udc.chatconnect.view.widget.Description
```

```
import com.udc.chatconnect.view.widget.PText
import com.udc.chatconnect.view.widget.Title


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun AppInfoView(back: () -> Unit) {
    Scaffold(topBar = {
        TopAppBar(title = {
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                Spacer(modifier = Modifier.size(0.dp))
                Text(
                    text = "App Info",
                    fontFamily = manrope,
                    fontWeight = FontWeight.Bold
                )
                Spacer(modifier = Modifier.width(42.dp))
            }
        }, navigationIcon = {
            IconButton(onClick = back) {
                Icon(
                    imageVector = Icons.Filled.ArrowBack,
                    contentDescription = "Back button"
                )
            }
        })

    }) {
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.SpaceBetween
        ) {
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Spacer(modifier = Modifier.height(32.dp))
                Image(
                    painter = painterResource(id = R.drawable.chat_icon_2),
                    contentDescription = null,
                    colorFilter = ColorFilter.tint(
                        Primary
                    ),
                    modifier = Modifier.height(46.dp)
                )
                Title(title = "Chat Connect")
                Description(desc = "A Real-Time Chat And Communication App")
            }
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Spacer(modifier = Modifier.height(16.dp))
                Text(
                    text = " Team ID : NM2023TMID11706",
                    fontSize = 16.sp,
                    fontFamily = manrope,
                    fontWeight = FontWeight.Bold,
```

```
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(8.dp))
        Text(
            text = " Team Leader : KARTHIKEYAN G",
            fontSize = 14.sp,
            fontFamily = manrope,
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(32.dp))
        Text(
            text = " Team Members",
            fontSize = 16.sp,
            fontFamily = manrope,
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(8.dp))
        PText(text = "MONIKA K")
        PText(text = "VIGNESH K")
        PText(text = "SHANMUGANATHAN P")
    }
    PText(text = "Version 1.1")
  }
 }
}
```

AppInfo.kt

```
package com.udc.chatconnect.view

import android.annotation.SuppressLint
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
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.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.ColorFilter
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.udc.chatconnect.R
import com.udc.chatconnect.ui.theme.Primary
import com.udc.chatconnect.ui.theme.manrope
import com.udc.chatconnect.view.widget.Description
```

```
import com.udc.chatconnect.view.widget.PText
import com.udc.chatconnect.view.widget.Title


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun AppInfoView(back: () -> Unit) {
    Scaffold(topBar = {
        TopAppBar(title = {
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                Spacer(modifier = Modifier.size(0.dp))
                Text(
                    text = "App Info",
                    fontFamily = manrope,
                    fontWeight = FontWeight.Bold
                )
                Spacer(modifier = Modifier.width(42.dp))
            }
        }, navigationIcon = {
            IconButton(onClick = back) {
                Icon(
                    imageVector = Icons.Filled.ArrowBack,
                    contentDescription = "Back button"
                )
            }
        })

    }) {
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.SpaceBetween
        ) {
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Spacer(modifier = Modifier.height(32.dp))
                Image(
                    painter = painterResource(id = R.drawable.chat_icon_2),
                    contentDescription = null,
                    colorFilter = ColorFilter.tint(
                        Primary
                    ),
                    modifier = Modifier.height(46.dp)
                )
                Title(title = "Chat Connect")
                Description(desc = "A Real-Time Chat And Communication App")
            }
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Spacer(modifier = Modifier.height(16.dp))
                Text(
                    text = " Team ID : NM2023TMID11706",
                    fontSize = 16.sp,
                    fontFamily = manrope,
                    fontWeight = FontWeight.Bold,
```

```
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(8.dp))
        Text(
            text = " Team Leader : KARTHIKEYAN G",
            fontSize = 14.sp,
            fontFamily = manrope,
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(32.dp))
        Text(
            text = " Team Members",
            fontSize = 16.sp,
            fontFamily = manrope,
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center
        )
        Spacer(modifier = Modifier.height(8.dp))
        PText(text = "MONIKA K")
        PText(text = "VIGNESH K")
        PText(text = "SHANMUGANATHAN P")
    }
    PText(text = "Version 1.1")
  }
 }
}
```