PROJECT REPORT TEMPLATE

1. INTRODUCTION

1.1 Overview

The project is a real-time group chat app that allows users to communicate with each other in real-time through a group chat. The app is designed to enhance collaboration and communication among users, especially in a team or group setting.

The app is built using Firebase, a cloud-based platform that provides scalable, reliable, and secure infrastructure for real-time apps. Firebase provides real-time synchronization of data, which means that users can see updates and messages instantly without the need to refresh the app. It also provides built-in security features, such as authentication and authorization, which can help protect user data and prevent unauthorized access to the app.

We created a chat app that allows users to chat with a group using Firebase. Firebase is a popular backend service that provides features such as real-time database, user authentication, and hosting.

users can join a group chat and send messages to other members in real-time. The messages are stored in Firebase's real-time database, which allows them to be synchronized across all users in the group. This means that when a user sends a message, all other users in the group will receive it immediately.

app does not have any special features like profile pictures or other advanced functionalities that are commonly found in messaging apps like WhatsApp. However, the core functionality of your app - group messaging - is still a valuable feature that can be useful for communicating with friends, family, or colleagues.

Security and Privacy: The Chat Connect app prioritizes user security and privacy, offering end-to-end encryption for messages and calls to protect user data. Users can also manage their privacy settings and control their information within the app.

Overall, the Chat Connect app offers a comprehensive communication solution with messaging, voice and video calls, file sharing, user profiles, search and discover, notifications, and robust security features. It facilitates seamless communication and connections with others, making it a versatile app for personal and professional use.

Firebase is a comprehensive and powerful mobile and web application development platform provided by Google. It offers a wide range of tools and services that developers can use to build scalable, robust, and feature-rich applications. Here's an overview of some of the key features of Firebase:

1.1 PURPOSE

project is actual occasion group chat app that admits consumers to correspond accompanying each.one in legitimate opportunity through a group chat. The app is devised to reinforce cooperation and ideas between consumers, exceptionally in a group or group background The app is erected utilizing Firebase, a cloud located terrace that determines climbable, trustworthy, and secure foundation real-occasion apps Firebase specifies absolute-period synchrony of dossier, that method that consumers can visualize renovates and ideas immediately outside the need to replenish the app. It too supports included freedom visage, in the way that confirmation and permission, that can help preserve consumer dossier and forbid unconstitutional approach to the app We constituted a chat app that admits consumers to chat accompanying a group utilizing Firebase Fire base is a common backend duty that supplies appearance in the way that certain-occasion table, consumer confirmation, and accommodating consumers can touch a group chat and transmit ideas to added appendages in certain period.

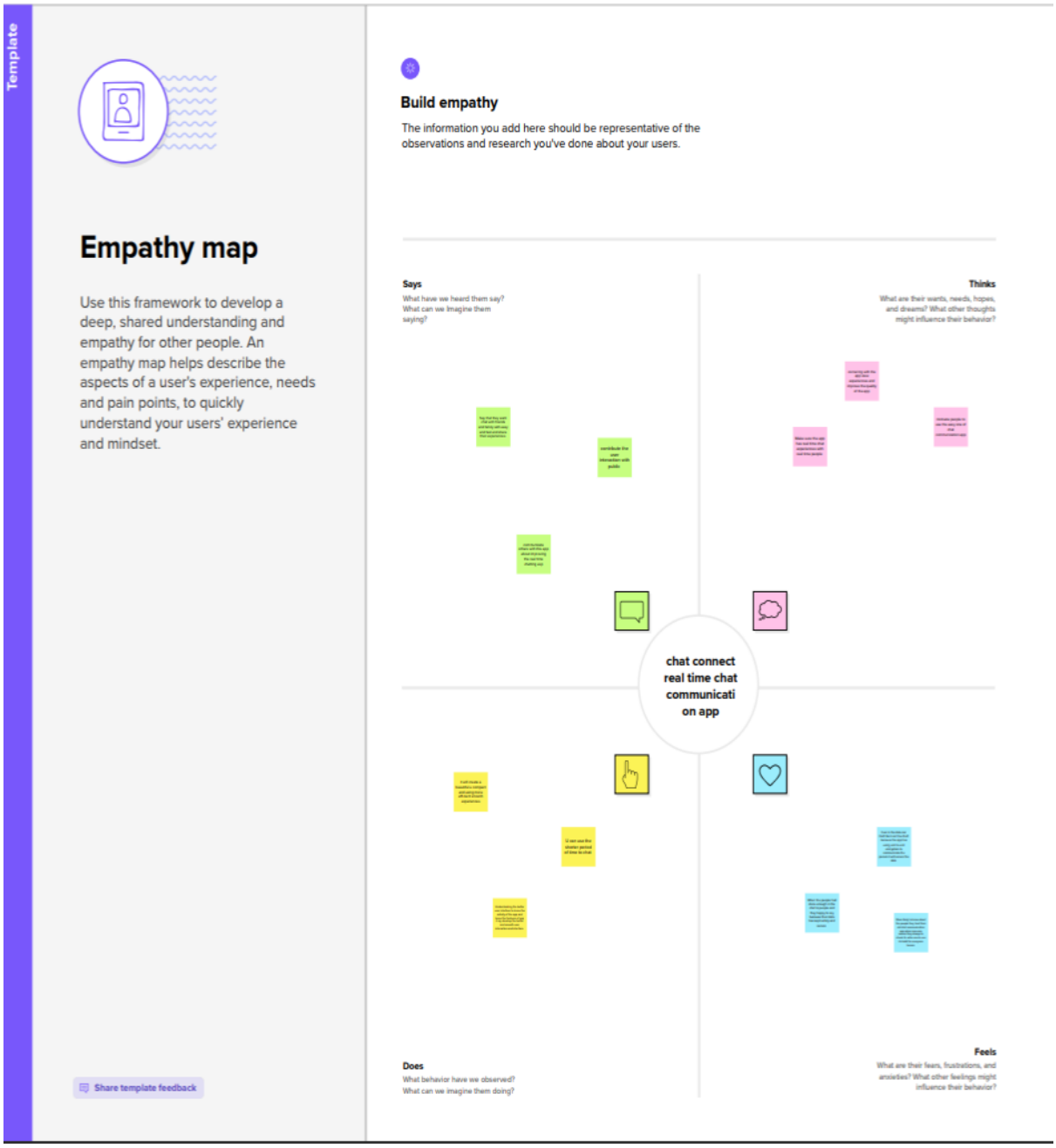
The ideas are stocked in Firebase's physical opportunity table, that admits bureaucracy expected coordinated across all consumers in the group. This resources that when a consumer sends a meaning, all different consumers in the group will sustain it shortly app does not have some distinguished facial characteristics like sketch pictures or different progressive functionalities that are usually about to foreshadow apps like WhatsApp. However, the gist r of capabilities of your app - group to foreshadow - is still a valuable feature that The maybe beneficial for corresponding accompanying companions, kin, or associates Security and Privacy: The Chat Connect a consumer protection and solitude, contribution end-to-end encryption for ideas and calls to defend consumer dossier. Users can too survive their solitude scenes and control their facts inside the app

Overall, the Chat Connect app offers inclusive ideas answer accompanying to foreshadow, voice and television calls, file giving, consumer characterizations, search and find, announcements, and strong safety face. It speeds smooth ideas and networks accompanying possible choice, making it a flexible app for private and professional use.

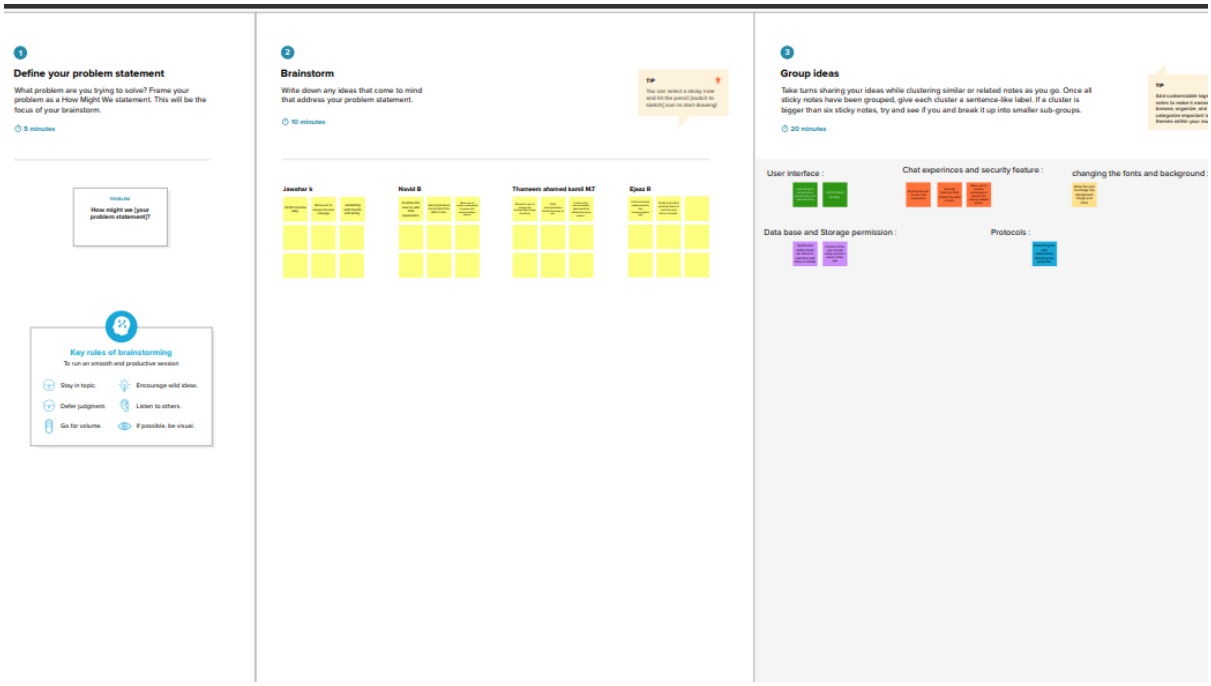
Firebase is inclusive and strong movable and netting use growth policy given by Google. It offers off course range of finishes and aids that builders can use to build ascendable, strong, and feature-rich requests.

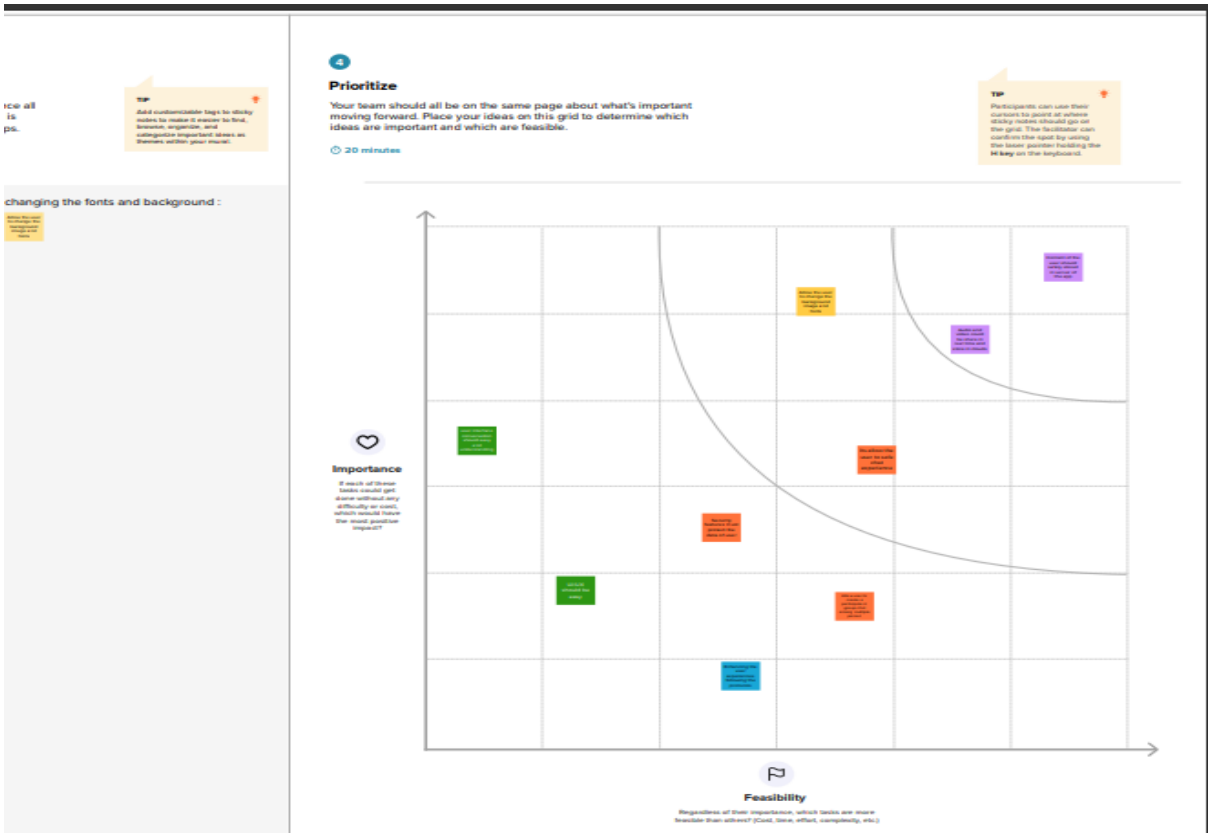
2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

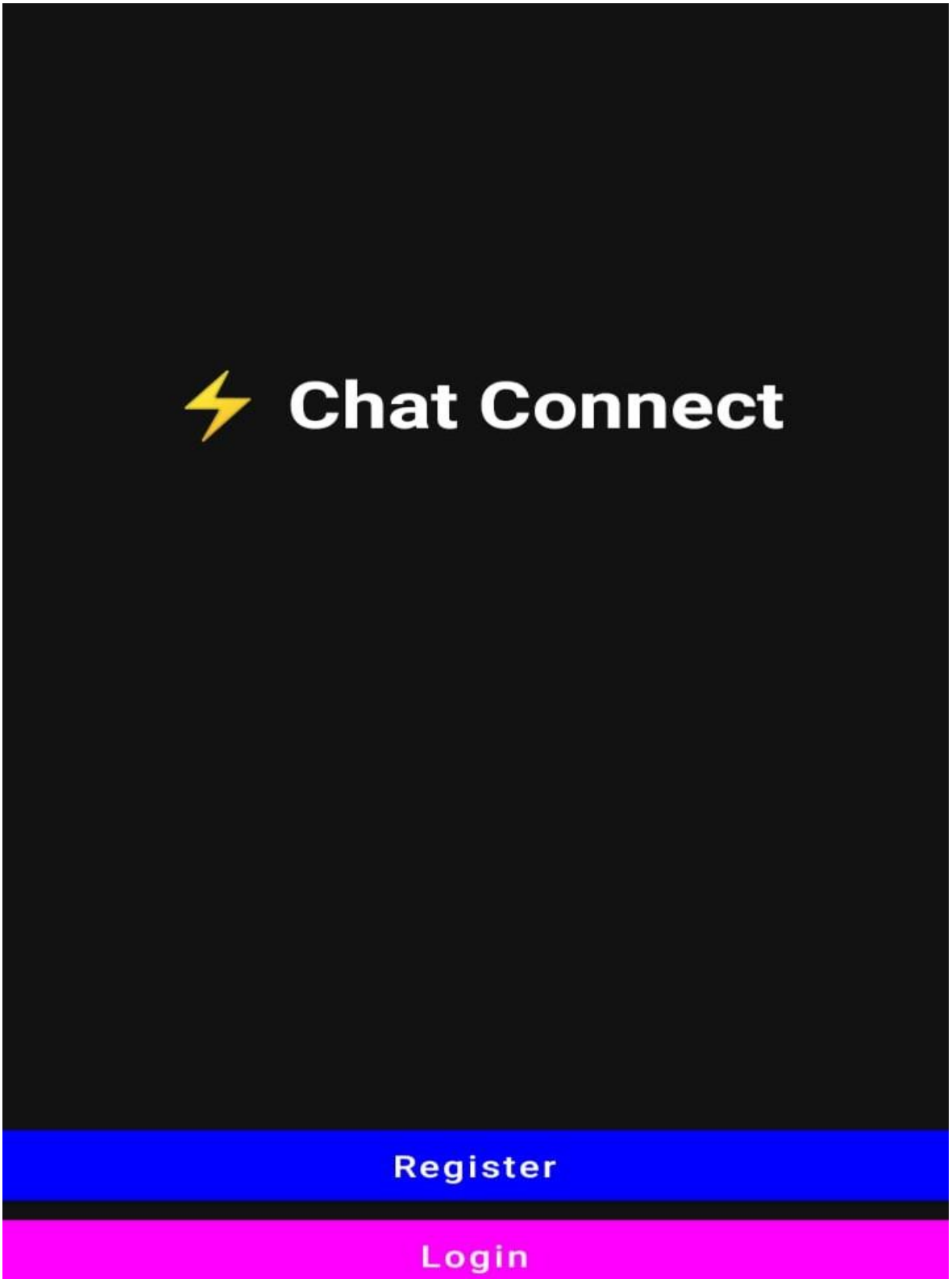


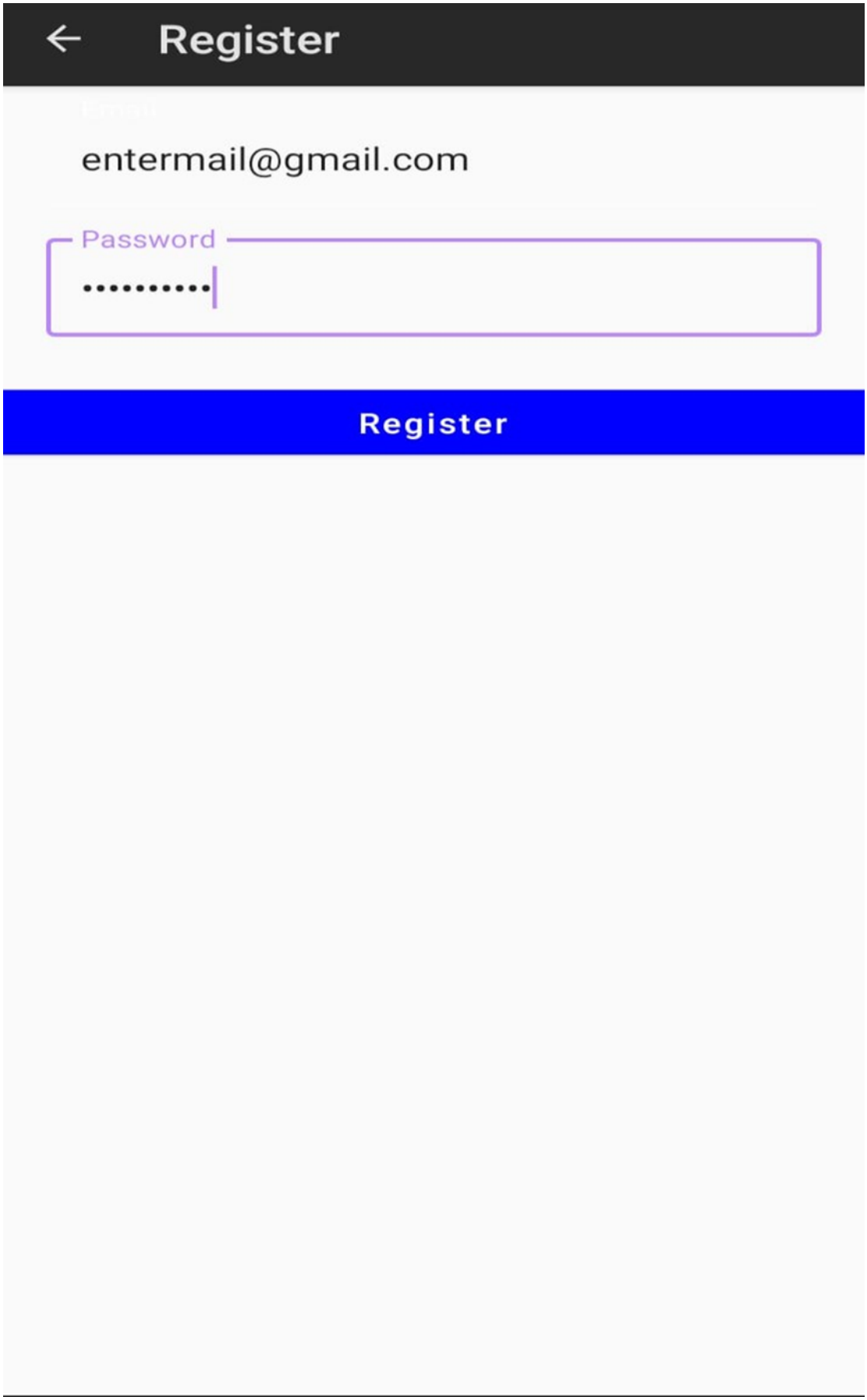
2.2 IDEATION & BRAINSTROMING MAP

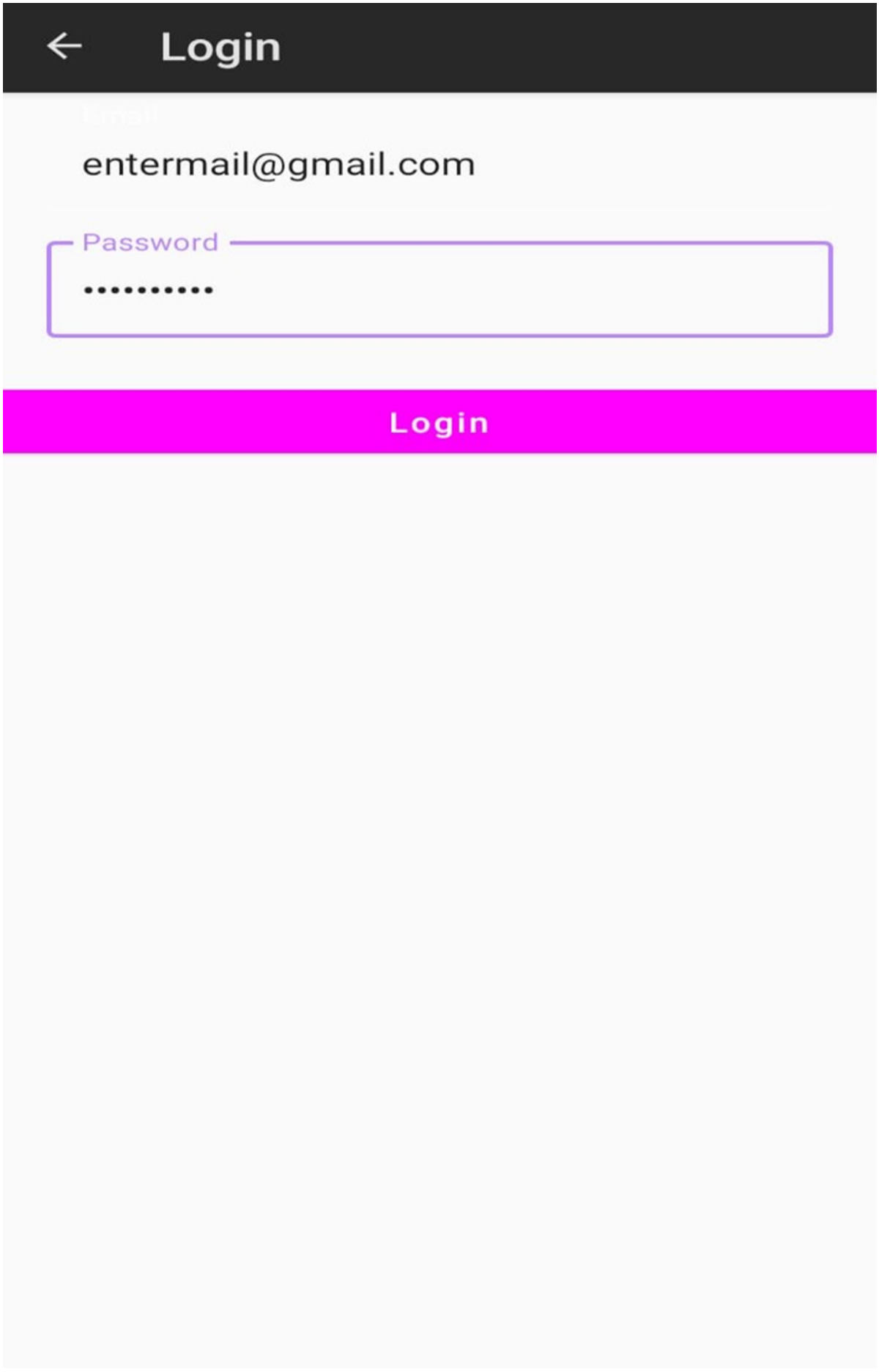


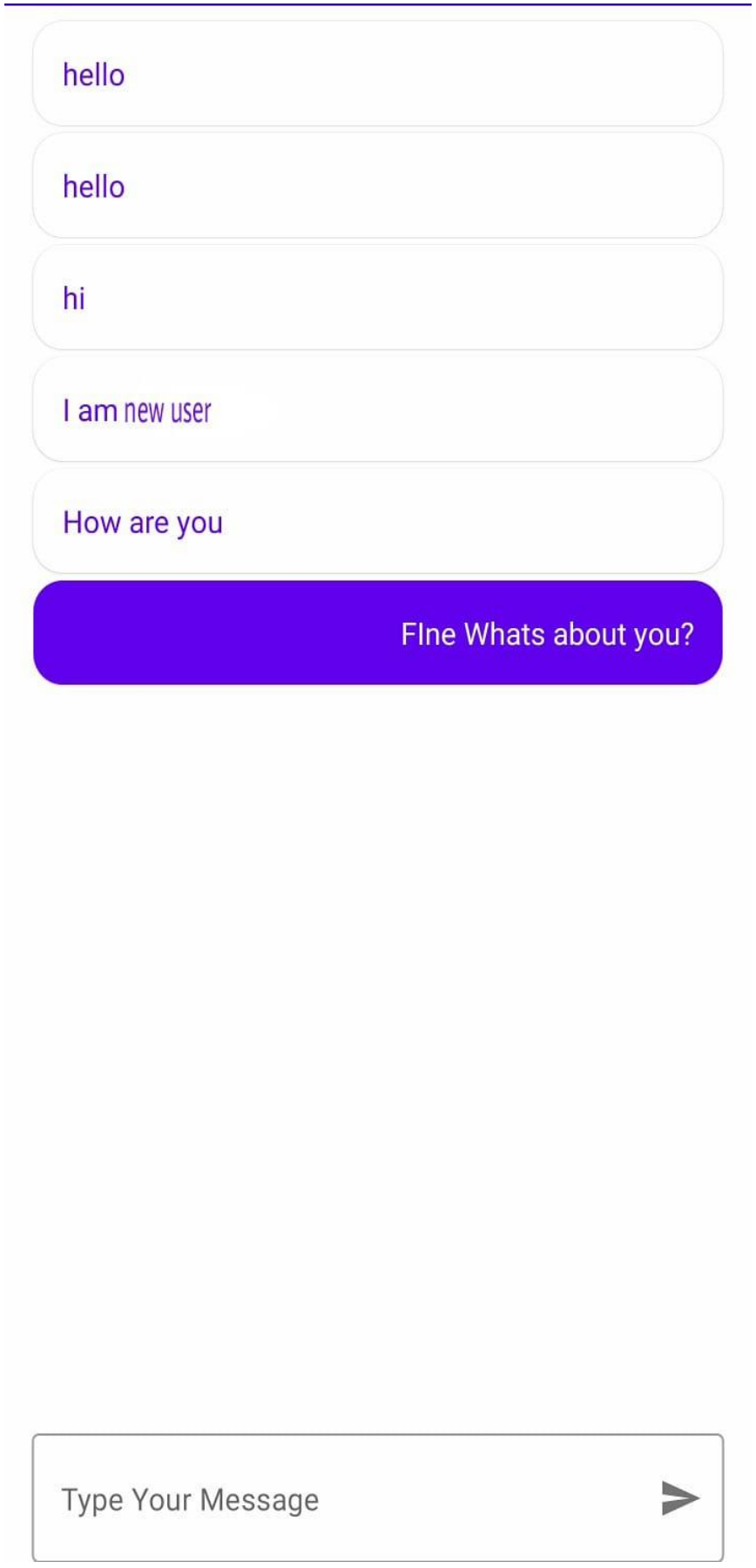


3.RESULT




Register
entermail@gmail.com
Password
Register


Login
entermail@gmail.com
Password
Login


hello
hello
hi
I am new user
How are you
FIne Whats about you?
Type Your Message

4. ADVANATGES & DISADVANTAGES

4.1 Advantages of chat-connect

Enhanced collaboration: Group chats allow multiple users to communicate with each other simultaneously, which can enhance collaboration and teamwork.

Efficient communication: Group chats can be an efficient way to communicate with a large number of people at once. Instead of sending individual messages to each person, users can simply send a message to the group.

Easy to manage: Group chats can be easier to manage than individual chats, as users can easily see all the messages and responses in one place.

[Scalability and reliability: Firebase is a cloud-based platform that can provide scalable and reliable infrastructure for a group chat app. This means that the app can handle a large number of users and messages without experiencing performance issues or downtime.

Firebase provides real-time synchronization of data, which means that users can see updates and messages instantly without the need to refresh the app. This can enhance the user experience and make the app more engaging.

Additionally, Firebase provides built-in security features, such as authentication and authorization, which can help protect user data and prevent unauthorized access to the app. This can help to address some of the security concerns associated with group chat apps.

Socialization: Group chatting can also be a fun way for people to socialize and connect with others who share similar interests.

Increased engagement: Group chats can increase engagement among users, as they can participate in conversations with multiple people and exchange ideas.

Increased transparency: Group chats can provide increased transparency among team members, as everyone has access to the same information and can stay up to date on the latest developments.

More efficient decision-making: Group chats can facilitate more efficient decision-making, as users can quickly share information, ask questions, and make decisionsEasier to coordinate: Group chats can be an effective way to coordinate activities and events, as all members can see the same information and respond quickly.

4.2 Disadvantages of chat-connect

Difficult to stay organized: Group chats can become disorganized quickly, especially if there are multiple conversations happening at the same time.

Lack of individual attention: Group chats can lead to a lack of individual attention, as messages can get lost or ignored in the group conversation.

Miscommunication: Group chats can also lead to miscommunication, as users may misinterpret messages or fail to convey their thoughts clearly.

Security concerns: Group chats can raise security concerns, especially if sensitive information is being shared. It's important to ensure that the app has adequate security measures in place to protect users' data.

Can be overwhelming: Group chats can be overwhelming for some users, especially if they are introverted or prefer one-on-one communication. It's important to offer users the option to participate in group chats or to communicate individually if they prefer.

Lack of privacy: Group chats are visible to all members of the group, which means that there is no privacy for individual conversations.

Distractions: Group chats can be distracting, with frequent notifications and messages from multiple people

.

Information overload: Group chats can also lead to information overload, as users may receive a large number of messages that can be difficult to keep track of.

Difficult to maintain focus: It can be challenging to maintain focus in a group chat, especially if multiple conversations are happening simultaneously.

5.APPLICATION

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, your real-time group chat app is a valuable tool for communication and collaboration among users, especially in a team or group setting.

Built using Firebase, the app provides scalable, reliable, and secure infrastructure for real-time communication, with features such as real-time synchronization of data, push notifications, and built-in security measures. This ensures that users can communicate and collaborate with confidence, knowing that their data is protected and their messages are delivered in real-time.

Overall, the chat connect app provides a convenient and efficient way for users to communicate and collaborate, helping to enhance productivity and engagement among team members. As the workplace becomes increasingly remote and teams become more dispersed, your app provides an essential tool for staying connected and working together effectively in real-time.

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, ands 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

7.1 Chat-connect can meet the future development & update

Here are some potential enchantments that could be made in the future for a real-time group chat app:

AI-powered language translation: Using artificial intelligence and natural language processing, the app could automatically translate messages in real-time to different languages, allowing users from different regions and language backgrounds to communicate more easily.

Augmented reality (AR) chat experiences: With advancements in AR technology, the app could allow users to have interactive and immersive chat experiences using AR, such as sending AR stickers, emojis, or animations that appear in the user's physical environment during video chat sessions.

Voice recognition and voice commands: The app could incorporate voice recognition technology, allowing users to send messages, initiate actions, or navigate the app using voice commands, which could be especially useful in hands-free or on-the-go situations.

Emotional analysis and sentiment tracking: The app could use machine learning algorithms to analyze the emotions and sentiment of messages, providing insights into the overall mood and sentiment of the group chat in real-time. This could help users gauge the tone of the conversation and facilitate more effective communication.

Advanced message scheduling: The app could allow users to schedule messages to be sent at a specific time or date in the future, which could be useful for planning events, reminders, or announcements within the group chat.

Smart notifications: The app could use intelligent algorithms to prioritize and filter notifications based on the user's preferences, importance of the message, or relevance to the user, reducing notification overload and improving the user's chat experience.

Personalized chatbots: The app could include personalized chatbots that can interact with users in a conversational manner, providing helpful information, answering questions, or assisting with tasks within the group chat.

Enhanced security features: Future enchantments could include advanced security measures, such as end-to-end encryption for all messages, biometric authentication for accessing the app, or blockchain-based solutions for data privacy and security.

Virtual reality (VR) integration: As VR technology becomes more widespread, the app could integrate VR capabilities, allowing users to have virtual chat experiences in immersive virtual environments, creating unique and interactive group chat experiences.

Social media integrations: The app could integrate with popular social media platforms, allowing users to share content, profiles, or group chat invitations on their social media accounts, expanding the reach and engagement of the group chat.

8.APPENDIX

8.1 Source code of the chat-connect

MainActivity.kt

```
package com.example.chatconnect
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.chatconnect
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.example.chatconnect.nav.Action
import com.example.chatconnect.nav.Destination.AuthenticationOption
```

```
import com.example.chatconnect.nav.Destination.Home
import com.example.chatconnect.nav.Destination.Login
import com.example.chatconnect.nav.Destination.Register
import com.example.chatconnect.ui.theme.ChatConnectTheme
import com.example.chatconnect.view.AuthenticationView
import com.example.chatconnect.view.home.HomeView
import com.example.chatconnect.view.login.LoginView
import com.example.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.home,
                back = actions.navigateBack
            )
        }
        composable(Login) {
            LoginView(
                home = actions.home,
                back = actions.navigateBack
            )
        }
        composable(Home) {
            HomeView()
        }
    }
  }
}
```

Navigation.kt

```
package com.example.chatconnect.nav
import androidx.navigation.NavHostController
import com.example.chatconnect.nav.Destination.Home
import com.example.chatconnect.nav.Destination.Login
import com.example.chatconnect.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.chatconnect
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.chatconnect.view
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import com.example.chatconnect.ui.theme.ChatConnectTheme

@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {
    ChatConnectTheme {
        // A surface container using the 'background' color from the theme
```

```
        Surface(color = MaterialTheme.colors.background) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Bottom
            ) {
                Title(title = "⚡ Chat Connect")
                Buttons(title = "Register", onClick = register, backgroundColor = Color.Blue)
                Buttons(title = "Login", onClick = login, backgroundColor = Color.Magenta)
            }
        }
    }
}
```

Widgets.kt

```
package com.example.chatconnect.view
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
```

```
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.chatconnect.Constants

@Composable
fun Title(title: String) {
    Text(
        text = title,
        fontSize = 30.sp,
        fontWeight = FontWeight.Bold,
        modifier = Modifier.fillMaxHeight(0.5f)
    )
}

// Different set of buttons in this page
@Composable
fun Buttons(title: String, onClick: () -> Unit, backgroundColor: Color) {
    Button(
        onClick = onClick,
        colors = ButtonDefaults.buttonColors(
            backgroundColor = backgroundColor,
            contentColor = Color.White
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = RoundedCornerShape(0),
    ) {
        Text(
```

```
            text = title
        )
    }
}

@Composable
fun Appbar(title: String, action: () -> Unit) {
    TopAppBar(
        title = {
            Text(text = title)
        },
        navigationIcon = {
            IconButton(
                onClick = action
            ) {
                Icon(
                    imageVector = Icons.Filled.ArrowBack,
                    contentDescription = "Back button"
                )
            }
        }
    )
}

@Composable
fun TextFormField(value: String, onValueChange: (String) -> Unit, label: String, keyboardType: KeyboardType, visualTransformation: VisualTransformation) {
    OutlinedTextField(
        value = value,
        onValueChange = onValueChange,
        label = {
```

```
            Text(
                label
            )
        },
        maxLines = 1,
        modifier = Modifier
            .padding(horizontal = 20.dp, vertical = 5.dp)
            .fillMaxWidth(),
        keyboardOptions = KeyboardOptions(
            keyboardType = keyboardType
        ),
        singleLine = true,
        visualTransformation = visualTransformation
    )
}

@Composable
fun SingleMessage(message: String, isCurrentUser: Boolean) {
    Card(
        shape = RoundedCornerShape(16.dp),
        backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else Color.White
    ) {
        Text(
            text = message,
            textAlign =
            if (isCurrentUser)
                TextAlign.End
            else
                TextAlign.Start,
            modifier = Modifier.fillMaxWidth().padding(16.dp),
```

```
            color = if (!isCurrentUser) MaterialTheme.colors.primary else Color.White
        )
    }
}
```

Home.kt

```
package com.example.chatconnect.view.home
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Send
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.chatconnect.Constants
import com.example.chatconnect.view.SingleMessage

@Composable
fun HomeView(
    homeViewModel: HomeViewModel = viewModel()
```

```
) {
    val message: String by homeViewModel.message.observeAsState(initial = "")
    val messages: List<Map<String, Any>> by homeViewModel.messages.observeAsState(
        initial = emptyList<Map<String, Any>>().toMutableList()
    )

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Bottom
    ) {
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .weight(weight = 0.85f, fill = true),
            contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
            verticalArrangement = Arrangement.spacedBy(4.dp),
            reverseLayout = true
        ) {
            items(messages) { message ->
                val isCurrentUser = message[Constants.IS_CURRENT_USER] as Boolean

                SingleMessage(
                    message = message[Constants.MESSAGE].toString(),
                    isCurrentUser = isCurrentUser
                )
            }
        }
        OutlinedTextField(
            value = message,
```

```
onValueChange = {
    homeViewModel.updateMessage(it)
},
label = {
    Text(
        "Type Your Message"
    )
},
maxLines = 1,
modifier = Modifier
    .padding(horizontal = 15.dp, vertical = 1.dp)
    .fillMaxWidth()
    .weight(weight = 0.09f, fill = true),
keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Text
),
singleLine = true,
trailingIcon = {
    IconButton(
        onClick = {
            homeViewModel.addMessage()
        }
    ) {
        Icon(
            imageVector = Icons.Default.Send,
            contentDescription = "Send Button",


        )
    }
}
```

```
        )
    }
}
```

HomeViewModel.kt

```
package com.example.chatconnect.view.home
import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.ktx.auth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import com.example.chatconnect.Constants
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

    fun updateMessage(message: String) {
        _message.value = 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 = ""
            }
        }
    }

    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)
        }
    }
    private fun updateMessages(list: MutableList<Map<String, Any>>) {
        _messages.value = list.asReversed()
    }
}
```

Login.kt

```
package com.example.chatconnect.view.login
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
```

```
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.chatconnect.view.Appbar
import com.example.chatconnect.view.Buttons
import com.example.chatconnect.view.TextFormField

@Composable
fun LoginView(
    home: () -> Unit,
    back: () -> Unit,
    loginViewModel: LoginViewModel = viewModel()
) {
    val email: String by loginViewModel.email.observeAsState("")
    val password: String by loginViewModel.password.observeAsState("")
    val loading: Boolean by loginViewModel.loading.observeAsState(initial = false)

    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier.fillMaxSize()
    ) {
        if (loading) {
            CircularProgressIndicator()
        }
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Top
        ) {
            Appbar(
```

```
            title = "Login",
            action = back
        )
        TextFormField(
            value = email,
            onValueChange = { loginViewModel.updateEmail(it) },
            label = "Email",
            keyboardType = KeyboardType.Email,
            visualTransformation = VisualTransformation.None
        )
        TextFormField(
            value = password,
            onValueChange = { loginViewModel.updatePassword(it) },
            label = "Password",
            keyboardType = KeyboardType.Password,
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(20.dp))
        Buttons(
            title = "Login",
            onClick = { loginViewModel.loginUser(home = home) },
            backgroundColor = Color.Magenta
        )
    }
  }
}
```

LoginViewModel.kt

```
package com.example.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
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
                }
        }
    }
}
```

Register.kt

```
package com.example.chatconnect.view.register
import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.chatconnect.view.Appbar
import com.example.chatconnect.view.Buttons
import com.example.chatconnect.view.TextFormField

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

    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier.fillMaxSize()
    ) {
        if (loading) {
            CircularProgressIndicator()
        }
        Column(
            modifier = Modifier.fillMaxSize(),
```

```
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Top
        ) {
            Appbar(
                title = "Register",
                action = back
            )
            TextFormField(
                value = email,
                onValueChange = { registerViewModel.updateEmail(it) },
                label = "Email",
                keyboardType = KeyboardType.Email,
                visualTransformation = VisualTransformation.None
            )
            TextFormField(
                value = password,
                onValueChange = { registerViewModel.updatePassword(it) },
                label = "Password",
                keyboardType = KeyboardType.Password,
                visualTransformation = PasswordVisualTransformation()
            )
            Spacer(modifier = Modifier.height(20.dp))
            Buttons(
                title = "Register",
                onClick = { registerViewModel.registerUser(home = home) },
                backgroundColor = Color.Blue
            )
        }
    }
}
```

RegisterViewModel.kt

```
package com.example.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
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 _loading = MutableLiveData(false)
    val loading: LiveData<Boolean> = _loading

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

```
    }

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

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

            _loading.value = true

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