

PROJECT TITLE

Chat Connect-A Real-Time Chat and Communication App

DESIGNED BY

Sri Paramakalyani College [code-123]

Alwarkurichi-627412

Department Of Computer Application

MENTOR

Smt.E.Jacqueline , M.C.A.,M.Phil,Asst.Prof.

Team

TEAM ID

NM2023TMID14924

TEAM LEADER

Manisha S

TEAM MEMBERS

Muthulakshmi Elakkiya G

Nanthini E

Sakthi Selvi S

PROJECT INDEX

1. Introduction

1.1 Overview

1.2 Purpose

2. Problem Definition And Design Thinking

2.1 Empathy Map

2.2 Ideation and Brainstorming Map

3. Result

4. Advantages and Disadvantages

5. Applications

6. Conclusion

7. Future Scope

8. Appendix

8.1 Source Code

1.INTRODUCTION

1.1 Overview

- Chat Connect is a real-time chat and communication app that allows users to connect with each other through instant messaging, voice calls, and video calls. It is designed to facilitate seamless communication between individuals or groups, making it ideal for personal, professional, or social interactions.
- A real-time chat communication app is a software application that enables users to communicate with each other in real-time. Unlike email or other forms of asynchronous communication, real-time chat apps allow users to exchange messages instantly and have a conversation in real-time, whether they are in the same room or on opposite sides of the world. Real-time chat apps have become increasingly popular in recent years, with many businesses and individuals relying on them for their day-to-day communication needs.
- Real-time chat apps offer several benefits over other communication methods. They are more convenient and faster than email, allowing users to communicate in real-time, which is especially useful for time-sensitive or urgent communication. These apps also support a range of features like group chat, file sharing, voice and video calls, and more, which make them a versatile tool for communication in a variety of settings. Real-time chat apps are also available on multiple devices, including smartphones, tablets, and desktops, making it easy for users to stay connected no matter where they are.
- Real-time chat apps are used in a wide range of settings, from casual social conversations to professional collaboration and customer support. They have become an essential tool for businesses to communicate with their teams and customers, especially in remote work environments. Real-time chat apps are also popular among individuals who want to stay connected with friends and family or meet new people with similar interests. Overall, real-time chat communication apps are an efficient and convenient way for people to communicate with one another in real-time, making them an essential tool in today's digital world. Here's an overview of Chat Connect's features:

➤ **Instant Messaging:**

Chat Connect provides a user-friendly interface for sending and receiving text messages in real-time. Users can exchange messages with individuals or groups, create chat rooms for specific topics, and share files such as images, documents, and videos.

- **Emojis and Stickers:**

Chat Connect includes a wide range of emojis, stickers, and GIFs to enhance the expression and emotions in conversations. Users can choose from a variety of emojis and stickers to add personality and fun to their messages.

- **Notifications and Alerts:**

Chat Connect provides notifications and alerts for new messages, voice calls, and video calls, ensuring that users are always aware of incoming communication even when the app is in the background. This feature helps users stay connected and responsive.

- **Multi-platform Access:**

Chat Connect is available on multiple platforms, including web browsers, mobile devices, and desktop applications, making it accessible and convenient for users to connect from different devices.

➤ Overall, Chat Connect is a comprehensive real-time chat and communication app that offers a range of features for seamless and convenient communication among individuals and groups. Whether for personal, professional, or social use, Chat Connect is designed to enhance communication and foster connections in real-time.

1.2 PURPOSE

- The purpose of Chat Connect, as a real-time chat and communication app, is to enable users to connect and communicate with each other in real-time through instant messaging, voice calls, and video calls. The app is designed to facilitate efficient and convenient communication, allowing users to exchange messages, have voice conversations, and conduct video calls with individuals or groups, all in real-time. These apps enable users to exchange messages, voice and video calls, and other types of information instantly, regardless of their location.
- Real-time chat communication apps are used for a variety of purposes, including social conversations, professional collaboration, and customer support. In social settings, real-time chat apps allow users to stay in touch with friends and family, share news and updates, and meet new people with similar interests. In professional settings, real-time chat apps are used for team communication, project management, and remote work collaboration. They allow team members to exchange information and feedback instantly, improving productivity and efficiency.
- Real-time chat communication apps are also commonly used for customer support, allowing businesses to communicate with their customers in real-time and provide quick and efficient assistance. They are an essential tool for e-commerce businesses that require instant communication with their customers to resolve issues and answer questions.

The primary purpose of Chat Connect includes:

- **Enhancing Communication:**

Chat Connect aims to provide users with a seamless and efficient way to communicate in real-time, enabling them to exchange messages, make voice calls, and conduct video calls. It aims to improve communication by offering a platform that is easy to use, reliable, and convenient for individuals or groups.

- **Fostering Connections:**

Chat Connect is designed to bring people together and foster connections. It enables users to connect with friends, family, colleagues, or

other communities of interest in real-time, regardless of their physical location. It provides a platform for users to build relationships, collaborate, and stay connected in a fast-paced digital world.

- **Enabling Collaboration:**

Chat Connect facilitates collaboration among individuals or groups through its real-time messaging, voice call, and video call features. It allows users to exchange ideas, share files, and work together on projects, making it useful for personal, professional, and team communication.

- **Enhancing Productivity:**

Chat Connect can be used for productive communication in professional settings, allowing teams to collaborate efficiently, share information, and make decisions in real-time. It can help streamline communication and reduce delays, leading to improved productivity and effectiveness in work-related contexts.

- **Enabling Social Interaction:**

Chat Connect also serves as a platform for social interaction, enabling users to connect with friends, family, or communities of interest, and engage in casual conversations, share updates, or simply stay connected in a social setting.

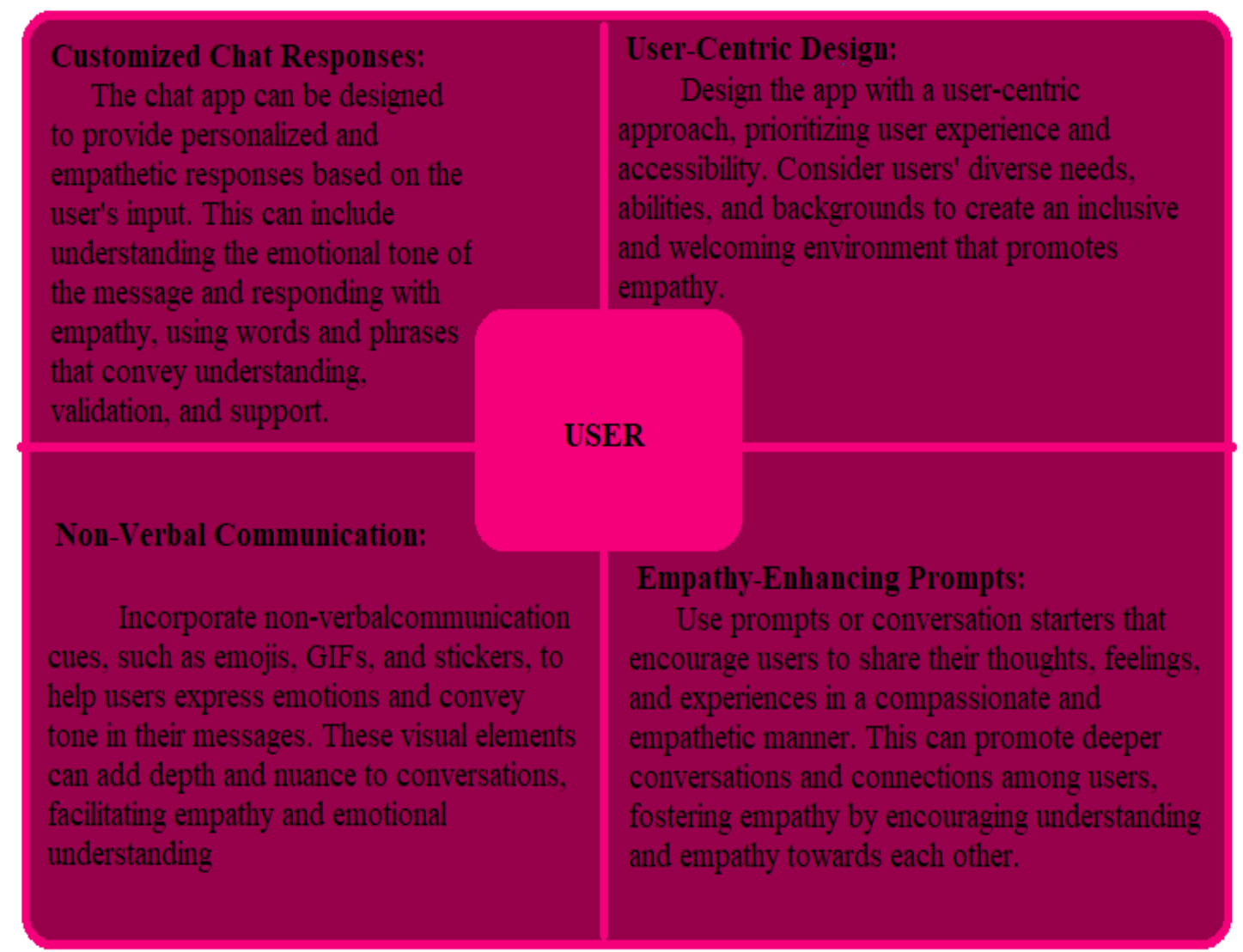
- **Providing Convenience and Flexibility:**

Chat Connect offers the convenience and flexibility of real-time communication across different devices, such as smartphones, tablets, and computers. It allows users to connect and communicate on the go or from different locations, making it accessible and adaptable to users' needs.

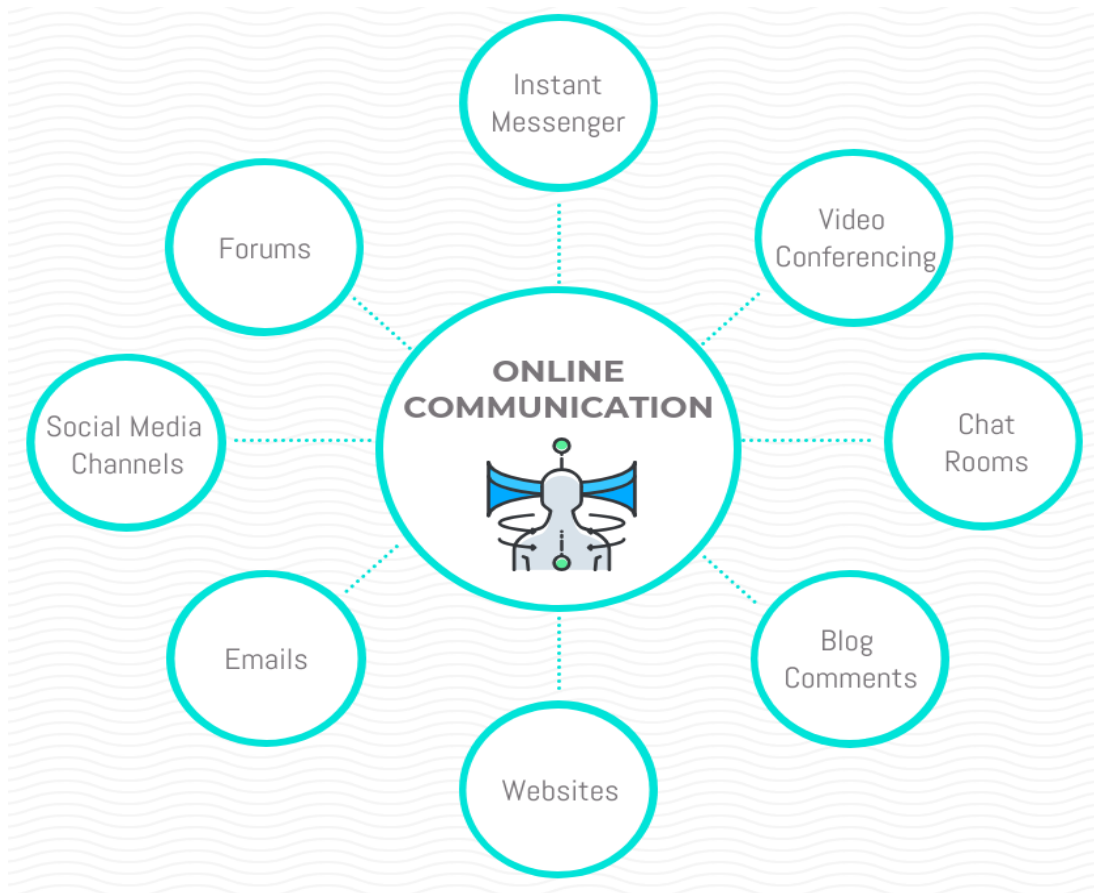
➤ In summary, the purpose of Chat Connect as a real-time chat and communication app is to provide users with a platform for efficient, convenient, and real-time communication, fostering connections, enabling collaboration, enhancing productivity, and facilitating social interaction.

2 . Problem Definition & Design Thinking

2.1 Empathy Map



2.2 Ideation and Brainstorming Map



3. RESULT

FIRST SCREEN:



Register

Login



REGISTER PAGE:



Register

Email

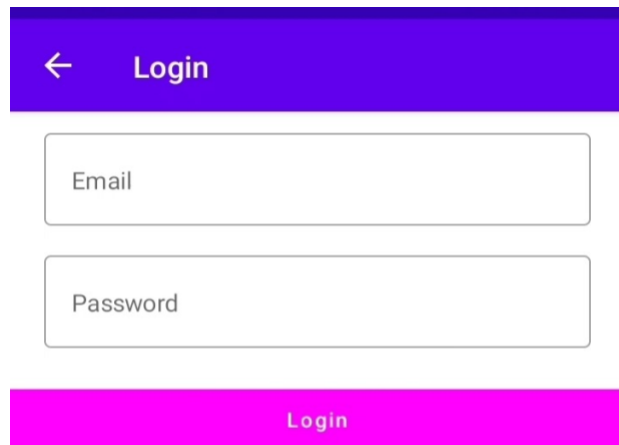
|

Password

Register



LOGIN PAGE:



The login page features a purple header bar with a white back arrow and the text "Login". Below the header are two white input fields with rounded corners, labeled "Email" and "Password". At the bottom is a solid red button with the text "Login".

← Login

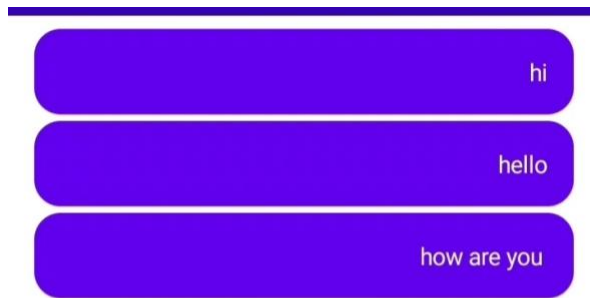
Email

Password

Login



MAIN PAGE:



4. ADVANTAGES AND DISADVANTAGES

4.1 Advantages

- Chat Connect, a real-time chat and communication app, offers several advantages for users:
- **Instant Communication:** With Chat Connect, users can communicate in real-time, enabling instant messaging and quick responses. This makes it

ideal for team collaborations, customer support, and staying connected with friends and family.

- **Convenience and Accessibility:**

Chat Connect allows users to communicate on-the-go, across different devices and platforms. It provides a convenient and accessible means of communication, allowing users to stay connected wherever they are.

- **Enhanced Collaboration:**

Chat Connect offers features such as group chats, file sharing, and screen sharing, which facilitate seamless collaboration among team members. It enables teams to work together efficiently, exchange ideas, and share information in real-time, leading to increased productivity.

- **Personalization and Customization:**

Chat Connect allows users to personalize their profiles, set status messages, and choose notification preferences. It offers customization options that enable users to tailor their chat experience according to their preferences and needs.

- **Security and Privacy:**

Chat Connect prioritizes security and privacy, offering end-to-end encryption, data protection, and user authentication features. It ensures that conversations and data exchanged within the app are secure and protected from unauthorized access.

- **Integration and Compatibility:**

Chat Connect integrates with other tools and platforms, allowing users to streamline their workflow and centralize their communication. It is compatible with various operating systems, devices, and networks, making it versatile and adaptable to different environments.

- **Enhanced User Experience:**

Chat Connect offers a user-friendly interface, intuitive navigation, and interactive features that enhance the overall user experience. It is designed to be user-centric, making it easy to use, visually appealing, and enjoyable for users.

- Overall, Chat Connect offers several advantages as a real-time chat and communication app, including instant communication, convenience, enhanced collaboration, personalization, security, integration, and an enhanced user experience.

4.2 Disadvantages

- While Chat Connect offers many advantages, it also has some potential disadvantages, including:

- **Dependence on Internet Connection:**

Chat Connect relies on an active internet connection for real-time communication. This means that users may face limitations or disruptions in communication if they have poor or no internet connectivity, which can be a disadvantage, especially in remote areas or during internet outages.

- **Information Overload:**

Real-time chat apps like Chat Connect can sometimes result in information overload, with constant messages, notifications, and updates coming in. This can be overwhelming and distracting, leading to decreased productivity and increased stress, if not managed effectively.

- **Miscommunication and Lack of Tone:**

Written messages in chat apps lack the nuances of tone, intonation, and body language that are present in face-to-face conversations.

This can sometimes result in miscommunication or misunderstandings, leading to confusion or conflicts among users.

- **Privacy Concerns:**

As with any online communication app, there may be privacy concerns with Chat Connect. Users need to be cautious about sharing sensitive or personal information, and ensure that their privacy settings are properly configured to protect their data from unauthorized access or breaches.

- **Distractions and Time Management:**

Real-time chat apps can sometimes lead to distractions, as users may receive constant notifications, messages, or alerts. This can impact productivity and time management, as users may spend excessive time on chat apps instead of focusing on their tasks or responsibilities.

- **Addiction and Overuse:**

Real-time chat apps can be addictive, leading to overuse or excessive reliance on them for communication. This can result in reduced face-to-face interactions, social isolation, and potential negative impacts on mental health and well-being.

- **Platform Dependence:**

Chat Connect may require users and their contacts to be on the same platform or app for effective communication. This can be a disadvantage if some contacts are not on the same app, resulting in limitations in communication options or the need to use multiple apps for different contacts.

- It's important to be aware of these potential disadvantages and manage them effectively to ensure a positive and productive experience while using Chat Connect or any other real-time chat and communication app.

5.APPLICATIONS

- Applications of chat connect A real time chat and communication app
- Chat Connect, as a real-time chat and communication app, can have various applications across different industries and use cases. Here are some examples:

- **Business Communication:**

Chat Connect can be used as an internal communication tool for businesses, allowing team members to collaborate, share ideas, and communicate in real-time. It can also facilitate communication with clients or customers, providing a direct channel for inquiries, support, and feedback.

- **Customer Service:**

Chat Connect can be integrated into customer service workflows, enabling businesses to provide real-time support to their customers. Users can chat with customer service representatives, receive prompt assistance, and resolve issues in real-time, improving customer satisfaction and retention.

- **Education:**

Chat Connect can be utilized as an online learning tool, allowing students and teachers to communicate in real-time for discussions, Q&A sessions, and group projects. It can also facilitate communication between educational institutions, such as for remote guest lectures or collaboration between schools.

- **Healthcare:**

Chat Connect can be used as a telehealth communication platform, enabling healthcare providers to communicate with patients in real-time for virtual consultations, remote monitoring, and telemedicine services. It can also facilitate communication among healthcare teams for efficient coordination and information sharing.

- **Events and Conferences:**

Chat Connect can be used as a communication tool for virtual events and conferences, allowing attendees to chat with speakers, network with other participants, and engage in real-time discussions. It can also be used for event announcements, updates, and scheduling.

- **Social Networking:**

Chat Connect can serve as a real-time messaging platform for social networking, allowing users to connect, chat, and share content with each other. It can include features such as group chats, multimedia sharing, and social media integration.

- **On-Demand Services:**

Chat Connect can be used as a communication tool for on-demand services such as food delivery, ride-sharing, or home services. It can enable real-time communication between service providers and customers for order tracking, updates, and support.

- **Gaming:**

Chat Connect can be integrated into online gaming platforms, allowing gamers to communicate in real-time during multiplayer games, coordinate strategies, and socialize. It can also include features such as in-game chat, voice chat, and group messaging.

- **Crisis Management:**

Chat Connect can be used as a communication tool during crisis situations, such as natural disasters or emergencies, to facilitate real-time communication among responders, volunteers, and affected individuals for coordination, updates, and support.

Overall, Chat Connect, as a real-time chat and communication app, has versatile applications in various industries and can facilitate efficient and convenient communication for different purposes.

6.CONCLUSION

In conclusion, the development of a real-time chat and communication app offers numerous benefits for users in various settings, such as personal, professional, and social contexts. By leveraging modern technologies, such as instant messaging, push notifications, and multimedia sharing, a real-time chat app can provide seamless and efficient communication experiences. Some of a real-time chat and communication app include:

- **Instantaneous communication:**

Users can send and receive messages in real-time, allowing for quick and efficient communication without delays. This can greatly enhance productivity and enable swift decision-making in professional settings.

- **Multi-platform accessibility:**

A well-designed chat app can be accessible across multiple platforms, such as smartphones, tablets, and desktop computers, making it convenient for users to communicate from any device with internet access.

- **Rich media sharing:**

Real-time chat apps often allow users to share multimedia content, such as images, videos, and documents, which can greatly enhance communication and collaboration among users.

- **Group communication:**

Real-time chat apps can support group chats, allowing users to communicate with multiple people simultaneously. This can be beneficial for team collaboration, social networking, or event planning.

- **Privacy and security:**

With advanced encryption and authentication features, real-time chat apps can provide a secure communication environment, protecting user data and ensuring privacy.

- **Customization and personalization:**

Real-time chat apps can be customized and personalized according to user preferences, allowing users to set their profiles, notifications, and preferences based on their needs and preferences.

- **Real-time customer support:**

Businesses can leverage real-time chat apps to provide instant customer support, enabling quick responses to customer inquiries and enhancing customer satisfaction.

➤ In conclusion, a well-designed real-time chat and communication app can greatly enhance communication, collaboration, and engagement among users in various contexts. By leveraging modern technologies and incorporating features such as instantaneous messaging, rich media sharing, group communication, and customization options, a real-time chat app can provide a seamless and efficient communication experience for users, making it a valuable tool in today's fast-paced digital world.

7.FUTURE SCOPE

➤ As we look into the future, the scope of Chat Connect, a real-time chat and communication app, is expected to expand significantly. Here are some potential future scenarios for the app:

- **Increased Global Adoption:**

With the advancement of technology and increasing connectivity, the global adoption of real-time chat and communication apps is

likely to rise. Chat Connect can become a popular choice for individuals, businesses, and communities around the world to communicate in real-time, share ideas, collaborate, and connect with each other.

- **Enhanced Features:**

As technology evolves, Chat Connect can incorporate advanced features such as AI-powered chatbots, real-time language translation, voice and video calling, augmented reality (AR) and virtual reality (VR) integration, and more. These features can enhance user experience and provide new ways for people to interact and communicate.

- **Diverse User Base:**

Chat Connect can cater to diverse user segments, including individuals, businesses, educational institutions, healthcare providers, government organizations, and more. The app can be customized to suit the unique needs of different users, making it a versatile tool for communication across various domains and industries.

- **Seamless Integration:**

Chat Connect can integrate with other communication platforms, productivity tools, and social media networks, allowing users to seamlessly communicate and collaborate across multiple channels. This can enhance the app's functionality and make it a central hub for communication and collaboration.

- **Data Privacy and Security:**

With increasing concerns about data privacy and security, Chat Connect can prioritize robust security measures, such as end-to-end encryption, multi-factor authentication, and data encryption at rest and in transit. This can instill trust among users and ensure their sensitive information is protected.

- **IoT and Wearables Integration:**

As the Internet of Things (IoT) and wearable devices become more prevalent, Chat Connect can integrate with these technologies, enabling users to communicate and interact using their smart devices, such as smartwatches, smart glasses, and other IoT devices. This can open up new possibilities for real-time communication in different contexts and scenarios.

- **Collaborative Workspaces:**

Chat Connect can evolve into a collaborative workspace, providing users with tools to work together in real-time, such as document sharing, task management, and project collaboration. This can make it a go-to app for remote work, team collaboration, and project management.

- **Personalized User Experience:**

Chat Connect can leverage machine learning and AI algorithms to analyze user behavior and preferences, and provide personalized recommendations and suggestions for communication and collaboration. This can enhance user engagement and satisfaction.

- **Integration with Virtual Assistants:**

As virtual assistants become more sophisticated, Chat Connect can integrate with these virtual assistants, allowing users to communicate and interact with them in real-time through the app. This can enable seamless communication with virtual assistants for tasks such as scheduling appointments, setting reminders, and getting information.

➤ Overall, the future scope of Chat Connect as a real-time chat and communication app is vast and promising, with potential for innovation, expansion, and customization to cater to the evolving needs of users across different industries and use cases.

8.APPENDIX

8.1 Source Code

MainActivity.kt

```
package com.project.pradyotprakash.flashchat
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.google.firebase.FirebaseApp

/**
 * The initial point of the application from where it gets started.
 *
 * Here we do all the initialization and other things which will be
 * required
 * thought out the application.
 */
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        FirebaseApp.initializeApp(this)
        setContent {
            NavComposeApp()
        }
    }
}
```

```
}
```

NavComposeApp.kt

```
package com.project.pradyotprakash.flashchat
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.google.firebase.auth.FirebaseAuth
import com.project.pradyotprakash.flashchat.nav.Action
import com.project.pradyotprakash.flashchat.nav.Destination.AuthenticationOption
import com.project.pradyotprakash.flashchat.nav.Destination.Home
import com.project.pradyotprakash.flashchat.nav.Destination.Login
import com.project.pradyotprakash.flashchat.nav.Destination.Register
import com.project.pradyotprakash.flashchat.ui.theme.FlashChatTheme
import com.project.pradyotprakash.flashchat.view.AuthenticationView
import com.project.pradyotprakash.flashchat.view.home.HomeView
import com.project.pradyotprakash.flashchat.view.login.LoginView
import com.project.pradyotprakash.flashchat.view.register.RegisterView
/**
 * The main Navigation composable which will handle all the navigation stack.
 */
@Composable
fun NavComposeApp() {
    val navController = rememberNavController()
    val actions = remember(navController) { Action(navController) }
    FlashChatTheme {
```

```

NavHost(
    navController = navController,
    startDestination =
    if (FirebaseAuth.getInstance().currentUser != null)
        Home
    else
        AuthenticationOption
) {
    composable(AuthenticationOption) {
        AuthenticationView(
            register = actions.register,
            login = actions.login
        )
    }
    composable(Register) {
        RegisterView(
            home = actions.home,
            back = actions.navigateBack
        )
    }
    composable(Login) {
        LoginView(
            home = actions.home,
            back = actions.navigateBack
        )
    }
    composable(Home) {

```



```

        HomeView()
    }
}
}
}

```

Constants.kt

```

package com.project.pradyotprakash.flashchat

object Constants {
    const val TAG = "flash-chat"
    const val MESSAGES = "messages"
    const val MESSAGE = "message"
    const val SENT_BY = "sent_by"
    const val SENT_ON = "sent_on"
    const val IS_CURRENT_USER = "is_current_user"
}

```

Navigation.kt

```

package com.project.pradyotprakash.flashchat.nav

import androidx.navigation.NavHostController
import com.project.pradyotprakash.flashchat.nav.Destination.Home
import com.project.pradyotprakash.flashchat.nav.Destination.Login
import com.project.pradyotprakash.flashchat.nav.Destination.Register

/**
 * A set of destination used in the whole application
 */

object Destination {

```

```

const val AuthenticationOption = "authenticationOption"
const val Register = "register"
const val Login = "login"
const val Home = "home"
}
/**
 * Set of routes which will be passed to different composable so that
 * the routes which are required can be taken.
 */
class Action(navController: NavHostController) {
    val home: () -> Unit = {
        navController.navigate(Home) {
            popUpTo(Login) {
                inclusive = true
            }
            popUpTo(Register) {
                inclusive = true
            }
        }
    }
    val login: () -> Unit = { navController.navigate(Login) }
    val register: () -> Unit = { navController.navigate(Register) }
    val navigateBack: () -> Unit = { navController.popBackStack() }
}

```

Color.kt

```
package com.project.pradyotprakash.flashchat.ui.theme

import androidx.compose.ui.graphics.Color

val Purple200 = Color(0xFFBB86FC)
val Purple500 = Color(0xFF6200EE)
val Purple700 = Color(0xFF3700B3)
val Teal200 = Color(0xFF03DAC5)
```

Shape.kt

```
package com.project.pradyotprakash.flashchat.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.project.pradyotprakash.flashchat.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
```

```
private val DarkColorPalette = darkColors(  
    primary = Purple200,  
    primaryVariant = Purple700,  
    secondary = Teal200  
)
```

```
private val LightColorPalette = lightColors(  
    primary = Purple500,  
    primaryVariant = Purple700,  
    secondary = Teal200  
)
```

```
@Composable
```

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

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

Type.kt

```
package com.project.pradyotprakash.flashchat.ui.theme
```

```
import androidx.compose.materialTypography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
```

```
/**
 * Set of Material typography styles to start with
 */
val Typography = Typography(
    body1 = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp
    )
)
```

Home.kt

```
package com.project.pradyotprakash.flashchat.view.home
```

```
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
```

```
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Send
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.project.pradyotprakash.flashchat.Constants
import com.project.pradyotprakash.flashchat.view.SingleMessage
```

```
/**
```

```
 * The home view which will contain all the code related to the view for
HOME.
```

```
 *
```

```
 * Here we will show the list of chat messages sent by user.
```

```
 * And also give an option to send a message and logout.
```

```
 */
```

```
@Composable
```

```
fun HomeView(
```

```
    homeViewModel: HomeViewModel = viewModel()
```

```
) {
```

```
val message: String by homeViewModel.message.observeAsState(initial =
    "")
```

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

```
Column(
    modifier = Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Bottom
) {
    LazyColumn(
        modifier = Modifier
            .fillMaxWidth()
            .weight(weight = 0.85f, fill = true),
        contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),
        verticalArrangement = Arrangement.spacedBy(4.dp),
        reverseLayout = true
    ) {
```

```
        items(messages) { message ->
            val isCurrentUser = message[Constants.IS_CURRENT_USER] as
Boolean
```

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

```

    }
}
OutlinedTextField(
    value = message,
    onChange = {
        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.project.pradyotprakash.flashchat.view.home
```

```

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.ktx.auth
import com.google.firebase.firestore.ktx.firestore
import com.google.firebase.ktx.Firebase
import com.project.pradyotprakash.flashchat.Constants
import java.lang.IllegalArgumentException

```

```
/**
```

```
 * Home view model which will handle all the logic related to HomeView
```

```
*/
```

```

class HomeViewModel : ViewModel() {
    init {

```

```

        getMessages()
    }

    private val _message = MutableLiveData("")
    val message: LiveData<String> = _message

    private var _messages = MutableLiveData(emptyList<Map<String,
Any>>()).toMutableList()
    val messages: LiveData<MutableList<Map<String, Any>>> = _messages

    /**
     * Update the message value as user types
     */
    fun updateMessage(message: String) {
        _message.value = message
    }

    /**
     * Send message
     */
    fun addMessage() {
        val message: String = _message.value ?: throw
IllegalArgumentException("message empty")
        if (message.isNotEmpty()) {
            Firebase.firestore.collection(Constants.MESSAGES).document().set(
                hashMapOf(
                    Constants.MESSAGE to message,
                    Constants.SENT_BY to Firebase.auth.currentUser?.uid,

```

```

        Constants.SENT_ON to System.currentTimeMillis()
    )
).addOnSuccessListener {
    _message.value = ""
}
}
}

/**
 * Get the messages
 */
private fun getMessages() {
    Firebase.firestore.collection(Constants.MESSAGES)
        .orderBy(Constants.SENT_ON)
        .addSnapshotListener { value, e ->
            if (e != null) {
                Log.w(Constants.TAG, "Listen failed.", e)
                return@addSnapshotListener
            }

            val list = emptyList<Map<String, Any>>().toMutableList()

            if (value != null) {
                for (doc in value) {
                    val data = doc.data
                    data[Constants.IS_CURRENT_USER] =
                        Firebase.auth.currentUser?.uid.toString() ==
data[Constants.SENT_BY].toString()

```

```

        list.add(data)
    }
}

updateMessages(list)
}
}

/**
 * Update the list after getting the details from firestore
 */
private fun updateMessages(list: MutableList<Map<String, Any>>) {
    _messages.value = list.asReversed()
}
}

```

AuthenticationOption.kt

```
package com.project.pradyotprakash.flashchat.view
```

```

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment

```

```

import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import com.project.pradyotprakash.flashchat.ui.theme.FlashChatTheme

/**
 * The authentication view which will give the user an option to choose between
 * login and register.
 */

@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {
    FlashChatTheme {
        // A surface container using the 'background' color from the theme
        Surface(color = MaterialTheme.colors.background) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Bottom
            ) {
                Title(title = "⚡ Chat Connect")
                Buttons(title = "Register", onClick = register, backgroundColor =
Color.Blue)
                Buttons(title = "Login", onClick = login, backgroundColor =
Color.Magenta)
            }
        }
    }
}

```

```
}  
}
```

Widgets.kt

```
package com.project.pradyotprakash.flashchat.view
```

```
import androidx.compose.foundation.layout.fillMaxHeight  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.foundation.text.KeyboardOptions  
import androidx.compose.material.*  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.ArrowBack  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.text.input.KeyboardType  
import androidx.compose.ui.text.input.VisualTransformation  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
import com.project.pradyotprakash.flashchat.Constants
```

```
/**
```

```
 * Set of widgets/views which will be used throughout the application.
```

```
 * This is used to increase the code usability.
```

```
*/
```

```
@Composable
```

```
fun Title(title: String) {
```

```
    Text(
```

```
        text = title,
```

```
        fontSize = 30.sp,
```

```
        fontWeight = FontWeight.Bold,
```

```
        modifier = Modifier.fillMaxHeight(0.5f)
```

```
    )
```

```
}
```

```
// Different set of buttons in this page
```

```
@Composable
```

```
fun Buttons(title: String, onClick: () -> Unit, backgroundColor: Color) {
```

```
    Button(
```

```
        onClick = onClick,
```

```
        colors = ButtonDefaults.buttonColors(
```

```
            backgroundColor = backgroundColor,
```

```
            contentColor = Color.White
```

```
        ),
```

```
        modifier = Modifier.fillMaxWidth(),
```

```
        shape = RoundedCornerShape(0),
```

```
    ) {
```

```
        Text(
```

```
            text = title
```

```
        )
```

```
}  
}
```

@Composable

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

@Composable

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



```

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

```

```

@Composable
fun SingleMessage(message: String, isCurrentUser: Boolean) {
    Card(
        shape = RoundedCornerShape(16.dp),
        backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else
        Color.White
    ) {
        Text(
            text = message,
            textAlign =

```

```

        if (isCurrentUser)
            TextAlign.End
        else
            TextAlign.Start,
        modifier = Modifier.fillMaxWidth().padding(16.dp),
        color = if (!isCurrentUser) MaterialTheme.colors.primary else
Color.White
    )
}
}

```

Login.kt

```

package com.project.pradyotprakash.flashchat.view.login

import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.project.pradyotprakash.flashchat.view.Appbar
import com.project.pradyotprakash.flashchat.view.Buttons

```

```
import com.project.pradyotprakash.flashchat.view.TextFormField
```

```
/**
```

```
 * The login view which will help the user to authenticate themselves and go to  
the
```

```
 * home screen to show and send messages to others.
```

```
 */
```

```
@Composable
```

```
fun LoginView(
```

```
    home: () -> Unit,
```

```
    back: () -> Unit,
```

```
    loginViewModel: LoginViewModel = viewModel()
```

```
) {
```

```
    val email: String by loginViewModel.email.observeAsState("")
```

```
    val password: String by loginViewModel.password.observeAsState("")
```

```
    val loading: Boolean by loginViewModel.loading.observeAsState(initial =  
false)
```

```
    Box(
```

```
        contentAlignment = Alignment.Center,
```

```
        modifier = Modifier.fillMaxSize()
```

```
    ) {
```

```
        if (loading) {
```

```
            CircularProgressIndicator()
```

```
        }
```

```
        Column(
```

```
            modifier = Modifier.fillMaxSize(),
```

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

```
    }  
  }  
}
```

LoginViewModel.kt

```
package com.project.pradyotprakash.flashchat.view.login
```

```
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.ktx.Firebase  
import java.lang.IllegalArgumentException  
  
/**  
 * View model for the login view.  
 */  
class LoginViewModel : ViewModel() {  
    private val auth: FirebaseAuth = Firebase.auth  
  
    private val _email = MutableLiveData("")  
    val email: LiveData<String> = _email  
  
    private val _password = MutableLiveData("")  
    val password: LiveData<String> = _password  
  
    private val _loading = MutableLiveData(false)
```

```

val loading: LiveData<Boolean> = _loading

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

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

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

        _loading.value = true

        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener {
                if (it.isSuccessful) {
                    home()
                }
            }
    }
}

```

```

        _loading.value = false
    }
}
}
}
}

```

Register.kt

```
package com.project.pradyotprakash.flashchat.view.register
```

```

import androidx.compose.foundation.layout.*
import androidx.compose.material.CircularProgressIndicator
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com.project.pradyotprakash.flashchat.view.Appbar
import com.project.pradyotprakash.flashchat.view.Buttons
import com.project.pradyotprakash.flashchat.view.TextFormField

/**

```

* The Register view which will be helpful for the user to register themselves into

* our database and go to the home screen to see and send messages.

*/

@Composable

fun RegisterView(

home: () -> Unit,

back: () -> Unit,

registerViewModel: RegisterViewModel = viewModel()

) {

val email: String by registerViewModel.email.observeAsState("")

val password: String by registerViewModel.password.observeAsState("")

val loading: Boolean by registerViewModel.loading.observeAsState(initial = false)

Box(

contentAlignment = Alignment.Center,

modifier = Modifier.fillMaxSize()

) {

if (loading) {

CircularProgressIndicator()

}

Column(

modifier = Modifier.fillMaxSize(),

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Top

) {

AppBar(

title = "Register",


```

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

```

RegisterViewModel.kt

```
package com.project.pradyotprakash.flashchat.view.register
```

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException

/**
 * View model for the login view.
 */
class RegisterViewModel : ViewModel() {
    private val auth: FirebaseAuth = Firebase.auth

    private val _email = MutableLiveData("")
    val email: LiveData<String> = _email

    private val _password = MutableLiveData("")
    val password: LiveData<String> = _password

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

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

```

    }

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

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

            _loading.value = true

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

```