Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

# ChatConnect - A Real-Time Chat And Communication App

*Developed By*

*Dhruv Umesh Sompura (20BIT0357)*

*Aditya Ranjan (20BIT0349)*

*Pranay Prakesh Rai (20BIT0129)*

*Gunnam Chandramouli (20BCE7296)*

# Table of Contents

# 1- INTRODUCTION

## 1.1 Overview

ChatConnect is an innovative social media chatting application designed to revolutionize the way people connect and communicate in the digital age. It offers a dynamic and interactive platform where individuals can engage in real-time conversations, share media content, and foster meaningful relationships. With its user-friendly interface and advanced features, ChatConnect aims to enhance social interactions, bridge geographical boundaries, and provide a seamless communication experience.

## 1.2 Purpose

The purpose of ChatConnect is to create a user-friendly and efficient chatting application that revolutionizes the way people connect and communicate. In today's fast-paced world, where geographical boundaries often limit physical interactions, ChatConnect aims to bridge that gap and foster meaningful connections. The app strives to enhance social interactions, empower individuals to express themselves, and create a sense of belonging in a digital landscape.

ChatConnect goes beyond mere messaging by providing a platform that encourages users to engage in conversations that matter. Whether it's connecting with old friends, making new acquaintances, or collaborating with colleagues, the app enables seamless communication, breaking down barriers and allowing users to share experiences, ideas, and emotions effortlessly.

Furthermore, ChatConnect is designed to cater to a wide range of interests and communities. It facilitates the creation of groups and communities based on shared hobbies, professional interests, or common goals. This fosters an environment where like-minded individuals can come together, exchange knowledge, and support one another, ultimately forming lasting relationships and communities.

In summary, ChatConnect is a user-friendly and efficient chatting application that aims to enhance social interactions, foster meaningful connections, and bridge geographical boundaries. It provides a platform where users can connect, communicate, and build communities, ultimately creating a more connected and inclusive digital world.

# 2- LITERATURE SURVEY

## 2.1 Existing problem

Social media and texting being an integral part of our lives not withstanding, there are multiple texting apps such as WhatsApp, Instagram direct and Facebook Messenger-that are widely used and loved. However, this does not in any way-mean that they are perfect. Some of the shortcomings of each app (along with a few of their advantages) is discussed below.

### 1- WhatsApp

**A) Synopsis-** WhatsApp (now owned by Meta) is one of the biggest social media texting apps used today. Over 12 billion messages have been sent through WhatsApp in India alone [GrabOn's analysis of the Indian Internet Market].

**B) Advantages-**

1. WhatsApp is a well-designed and simple-to-use platform for sharing info and messages.
2. WhatsApp has recently evolved into a fictitious e-learning platform. In addition to sharing notes and information, teachers can create learning videos or share YouTube videos in WhatsApp groups.
3. Personal information is much more secure because of WhatsApp's secure end-to-end encryption feature.
4. Allows global connection i.e a person living in Japan can talk to someone in the US with little to no difficulty.
5. Whatsapp is completely free and has no ads-making it not only ideal for communication but also for conducting business.

**C) Shortcomings-**

1. With unchecked messages running rampant-WhatsApp has become a means of spreading false information.
2. The chat between two people on WhatsApp can remain unrecorded and unsupervised due to the various security features the app offers, which is possibly one of its scariest aspects. Anyone can delete any messages sent back and forth between two people, from disappearing messages to the "delete for everyone" option.
3. No content censorship that can lead to financial fraud and scams running rampant.

## 2- **Instagram Direct**

**A) Synopsis**-In San Francisco, Kevin Systrom and Mike Krieger launched Instagram after initially intending to create a platform similar to Foursquare but eventually concentrating on photo sharing. The phrases "instant camera" and "telegram" are combined to form the name Instagram. The iOS software was made accessible through the iTunes Version Store on October 6, 2010, and the Android app on April 3, 2012. Instagram

**B) Advantages**

1- Instagram direct is low cost or mostly free.

2- Allowing global communication increases contacts and allows people to keep in touch, across the globe. Anyone who misses their former friends may look them up on Instagram, and company owners can even create profiles and request that people follow them. Business executives and regular people together can enhance their interaction in this way.

3- People frequently share their videos, photos, and business information. Individuals use this platform to share their posts, stories, and numerous other things to keep others informed about what they are doing.

4- Business now have more market prospects due to their ability to promote goods and study consumer demand via Instagram.

5- Instagram is steadily connected to other networks like Facebook and Twitter thereby increasing its already wide reach.

**C) Shortcomings**

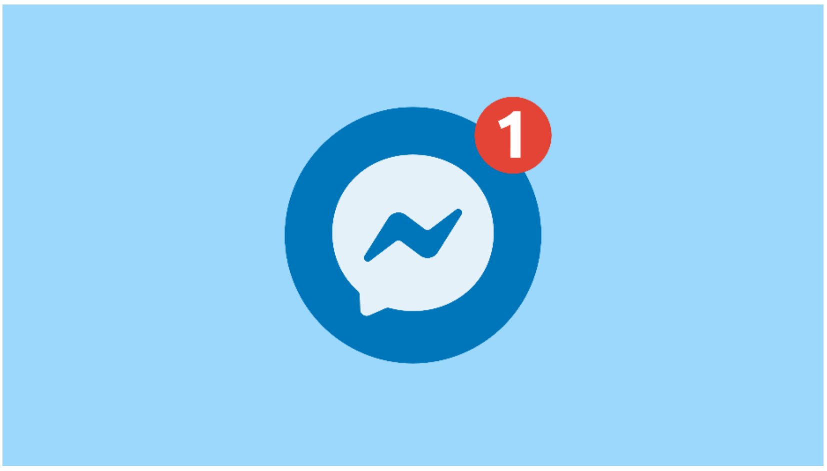
1- Instagram is built primarily for smartphones-many of the features are not available on computers.

2- Since others might use submitted photos for professional gain, they risk being stolen. This indicates that the image may be used for commercial gain without the user's permission.

3- If your account is public-there is no stopping malicious people from making untoward and creepy comments.

4- There is little to no censorship regarding what you can post.

## 3- Facebook Messenger



A) **Synopsis**- The Facebook mobile app gives you access to your timeline through an icon on your phone. Facebook Messenger app is a supplement app that allows you to send and receive messages via Facebook.

**B) Advantages**

1- You can send and receive messages more quickly via the app than if you needed to load the mobile browser.

2- The style of sending a message via Facebook Messenger is similar to sending a text message on your mobile phone. However, it's free. For people with mobile phones that charge for texting, this is a valuable feature.

3- You can quickly share files, images and links.

4- Easy phone calls. The app's audio connects you to either another individual or a group. This feature is especially handy for those calling loved ones outside of the United States.

**C) Shortcomings**

1- The terms and conditions include recording audio at any time. For many, this is a major privacy breach.

2- It's intrusive. Because the app is so similar to texting, it can be a distraction. Constant message notifications can keep you from working on more important matters.

3- Battery sucks. When activated, Facebook Messenger is always running in the background, which means it pulls from your battery life.

4- No easy alternative. If you use the Facebook mobile app, you are required to use Facebook Messenger to send messages via Facebook. This means you cannot read messages within the app if you have not installed Facebook Messenger. You will get the notification that you've received a message, but you won't be able to read it on your mobile phone.

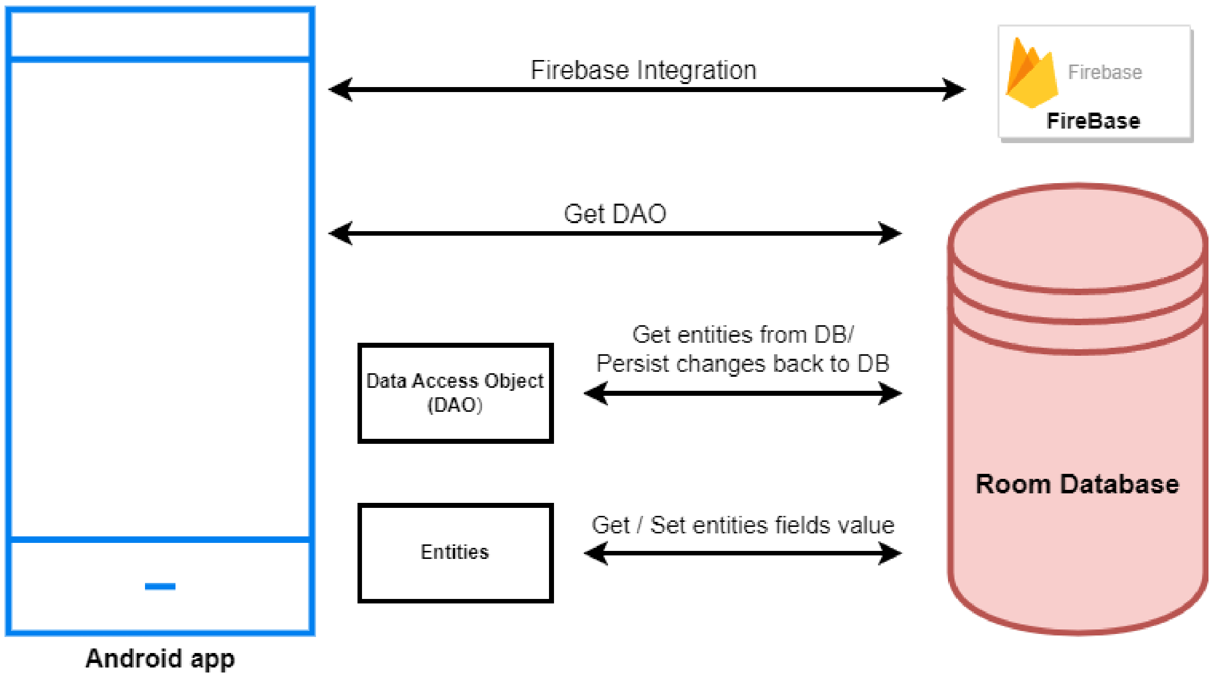
5- Storage. The Facebook app and Facebook Messenger use a combined 100+ MB of storage space on your mobile device.

## 2.2 Proposed solution

Our prototype app is a simple text/multimedia communication-based application. We will try to develop this app, keeping the major drawbacks of the afore-mentioned social media apps in mind.

# 3- THEORITICAL ANALYSIS

## 3.1 Block diagram



## 3.2 Hardware / Software designing

### Hardware Components:

1. Development Computer: A computer with sufficient processing power and memory to run the Android development tools smoothly.
2. Android Device: A physical Android device or an Android emulator to test the application during development.

### Software Components:

1. Android Studio: An Integrated Development Environment (IDE) specifically designed for Android app development. It provides tools for coding, debugging, and testing Android applications.
2. Java Development Kit (JDK): The JDK is required to develop Android applications using Java programming language.
3. Firebase Account: Sign up for a Firebase account to utilize Firebase services for real-time database, authentication, and cloud messaging.

4. Firebase SDK: Include the Firebase SDK in your Android project to integrate Firebase services into your application.
5. Android SDK: Install the Android Software Development Kit (SDK) to access the necessary libraries and tools for building Android applications.
6. Gradle: Gradle is a build automation tool used in Android Studio to manage dependencies and build the application.
7. Google Play Services: Google Play Services library should be included in your Android project to utilize Firebase services.

## 4- EXPERIMENTAL INVESTIGATIONS

### Reporting with WhatsApp

Review-The paper aims to examine the incentives that drive journalists to use WhatsApp and the consequences of this usage on their relationship with sources, both on a personal and professional level. It suggests that the utilization of WhatsApp may have implications for intimacy, trust, camaraderie, obtainability, and temporality among journalists who use the application.The paper also highlights that the observations made in the paper have important professional and ethical implications for journalists navigating today's media ecology, and indicate how technological and socio-professional aspects are intertwined.Based on the abstract, it appears that the paper provides insights into how WhatsApp is being used by journalists in Chilean newsrooms and sheds light on the impact of this usage on their newsgathering practices and relationships with sources. It also suggests that the findings have broader implications for the professional and ethical considerations of journalists in the current media landscape.

### Text Encryption in Android Chat Applications using Elliptical Curve Cryptography (ECC)

Review-The paper highlights the importance of information technology in human life, specifically in the realm of communication, with smartphones being the most commonly used communication technology today. The convenience of chat applications, such as the absence of message size limitations, popularity among younger demographics, and compatibility with mobile web, is mentioned. However, with the increasing demand for security in communication, the need for proper cryptography schemes to protect messages is emphasized. The paper aims to implement an Elliptic Curve Cryptography (ECC) algorithm to secure text messages in a messaging application for smartphones. The methodology proposed by Singh1 is adopted to create an Android chat application with end-to- end encryption. The abstract also mentions that experimental results, including the accuracy of received text messages, average encryption and decryption time, will be provided. Based on the abstract, it appears that the paper focuses on implementing ECC for securing text messages in a chat application and evaluating the performance of the developed application.

## Encryption Methods and Comparison of Popular Chat Applications

Review-The paper highlights the increasing security challenges in communication due to the rapid advancement of technology in today's global world. The need to protect confidential data, including photographs, videos, and sound recordings, is emphasized. The paper aims to propose a solution to this problem by implementing end-to-end encryption in chat applications to enable secure exchange of private information among users. The abstract mentions that the article presents a list of requirements for developing a secure chat application. Based on the abstract, it appears that the paper focuses on proposing a solution for ensuring privacy and security in chat applications through end-to-end encryption.

## Insularized Connectedness: Mobile Chat Applications and News Production

Review-This paper focuses on the role of mobile chat applications, such as WhatsApp, WeChat, LINE, Facebook Messenger, and others, in journalism, specifically in the context of recent political unrest in Hong Kong. The paper examines how these mobile chat apps have permeated journalism in Hong Kong, serving as tools for foreign correspondents to follow stories, identify sources, verify facts, and manage large flows of multimedia information in real-time. The study draws on 34 interviews conducted by the author with journalists who use mobile chat apps in their reporting, and explores the institutional, technological, and cultural factors at play.The paper adopts the concept of media logic and explores the impact of technology-involved social interactions on media work, while acknowledging the agency of users and audiences within the cultural context of Hong Kong. It argues that mobile chat apps have become hosts for a logic of connectedness and insularity in media work, leading to new forms of co- production in journalism.Overall, this paper offers insights into the role of mobile chat apps in journalism, particularly in the context of political unrest, and contributes to the understanding of how these technologies are shaping contemporary journalistic practices in Hong Kong.

## Android-Based Chat Application Using Firebase

Review-This paper proposes the development of a web-based Android chat application called BONJOUR, which aims to facilitate communication between users connected to the internet while preventing the sending of inappropriate messages. The paper mentions that there are various chat applications available in the market, with Facebook Messenger, QQ Mobile, and WhatsApp being widely used. The scope of the project includes the development and testing of features such as user registration, menu creation, client/server interface, and graphical user interfaces (GUIs) for user convenience. Additional testing will be conducted to further enhance the usability of the chat application.

## Developing a Multi-Purpose Chat Application for Mobile Distributed Systems on Android Platform

Review-The main objective of this project was to design and implement a multi-purpose chat application for mobile distributed systems that supports instant messaging, file sharing, and downloading files from a remote web server. The chat application is based on a peer-to-peer network, eliminating the need for a central server for communication among peers. The client side implementation of the application was done using Eclipse IDE with Android Development Tools (ADT) plugin and Java language, along with the peerdroid library. Additionally, a rendezvous peer, which serves as a gathering point for peers on the JXTA network, was implemented using Netbeans IDE and Java language.The results of the project demonstrate that it is possible for multiple peers connected on the JXTA network to communicate in real-time and share resources with each other. Users of the multi-purpose chat application were also able to download images from a remote web server and save them on their local secure digital (SD) card for future sharing with other peers on the network.Overall, the project successfully implemented a chat application based on peer-to-peer network architecture, allowing for real-time messaging, file sharing, and remote file downloading. The results highlight the feasibility and potential of such applications in mobile distributed systems. Further evaluation and testing may be conducted to assess the performance, scalability, and security aspects of the chat application.

## A secure chat application based on pure peer-to-peer architecture

Review-The focus of this study is to propose a chat application that utilizes a pure peer-to-peer architecture, eliminating the need for centralized or third-party elements. This approach aims to provide users with increased control and autonomy over their communication, while also enhancing security measures. The proposed chat application allows users to exchange text messages, as well as audio and video communications. Each user has their own database for peer profiles, and communication parties authenticate with each other before exchanging messages.The main contribution of this paper is the development of a state-of-the-art chat application that is designed with built-in security measures. By utilizing a pure peer-to-peer architecture, the proposed system aims to provide a secure and autonomous communication environment for users, without relying on centralized servers or third-party intermediaries. This approach may offer advantages in terms of user control, privacy, and security, compared to traditional chat applications that rely on centralized servers for user registration and buddy lists.

## Automated Chat Application Surveys Using Whatsapp: Evidence from Panel Surveys and a Mode Experiment

Review-The study presents a novel method of conducting automated surveys over WhatsApp, a widely used messaging service with global popularity. The advantages of using WhatsApp for surveys include low costs to respondents and researchers, ease of use for respondents familiar with WhatsApp chat features, and continued engagement with mobile populations. The method is tested through original panel surveys of refugees in Colombia and the U.S., demonstrating satisfactory response rates and retention over a nine-month follow-up period.

Additionally, a mode experiment comparing WhatsApp to short message service (SMS) and interactive voice response (IVR) surveys in Colombia reveals higher response rates for WhatsApp, indicating higher initial engagement and lower break-off compared to other methods. The study concludes by discussing the advantages and limitations of the WhatsApp method and providing documentation and a public code repository to support researchers and practitioners in applying the method in other contexts.

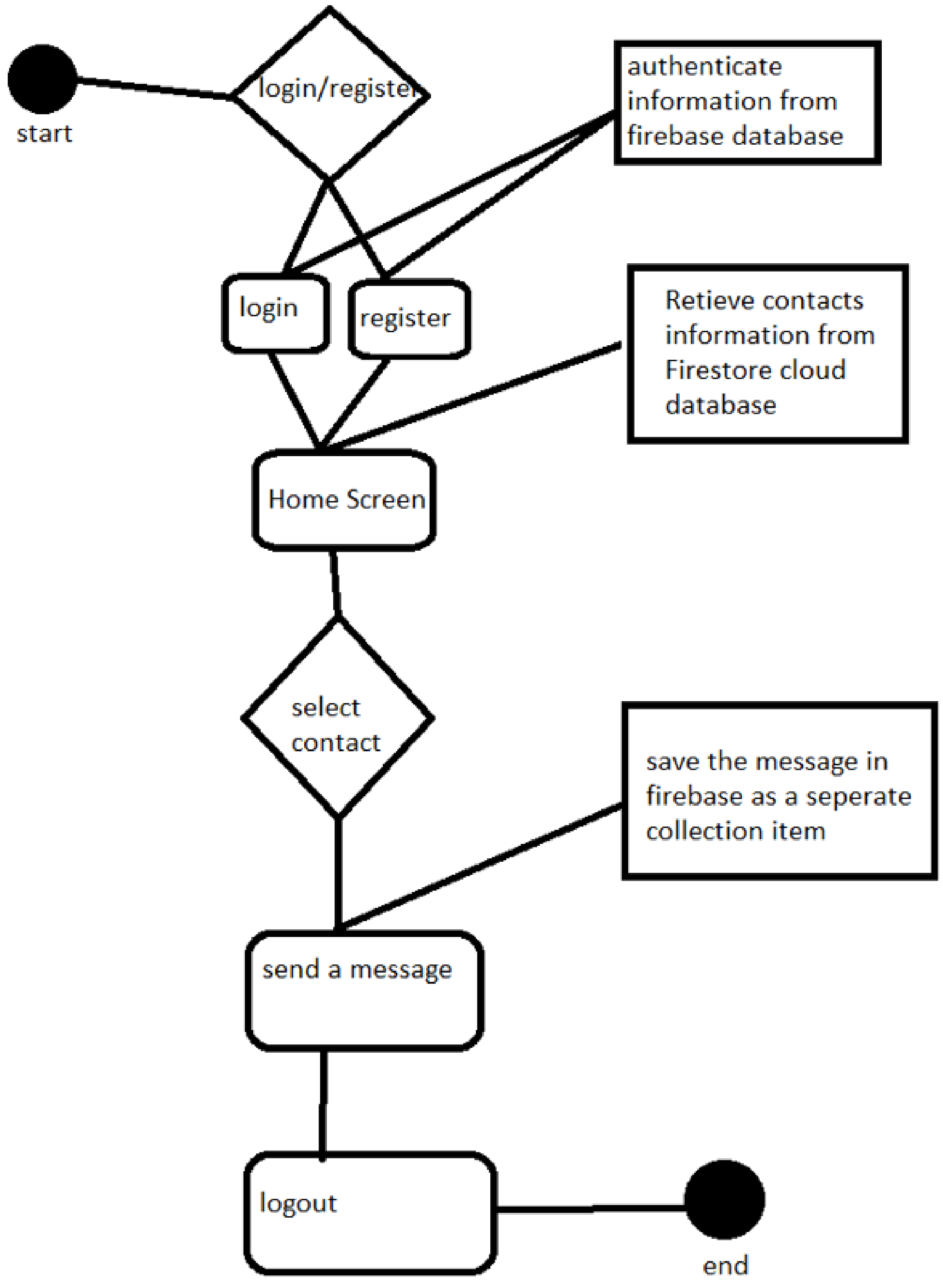
## Lie-Sensor: A Live Emotion Verifier or a Licensor for Chat Applications using Emotional Intelligence

Review-The paper proposes an emotion intelligent live detector that acts as an arbiter to verify the veracity of messages exchanged in chat apps. The detector uses facial expressions and text prediction to determine the emotions of chat app users and messages. The emotions are categorized into labels such as Happiness, Sadness, Surprise, and Hate. Convolutional Neural Network (CNN) using a miniXception model is deployed for emotion detection from facial expressions, and Support Vector Machine (SVM) natural language processing probability classifier is used for text prediction. SVM is selected as the best classifier after comparing with other classifiers such as Random Forest Classifier, Naive Bayes Classifier, and Logistic Regression, based on accuracy on the training dataset. The proposed method aims to detect fraudulent or bonafide messages in live chats, corroborate messages between users, and promote honest conversations by verifying the emotions and content of messages. The concept of veracity in research and development of innovative products is emphasized as essential, and the paper presents a novel approach using emotion AI for message verification in chat apps.

## Determining How Social Media Affects Learning English: An Investigation of Mobile Applications Instagram and Snap Chat in TESOL Classroom

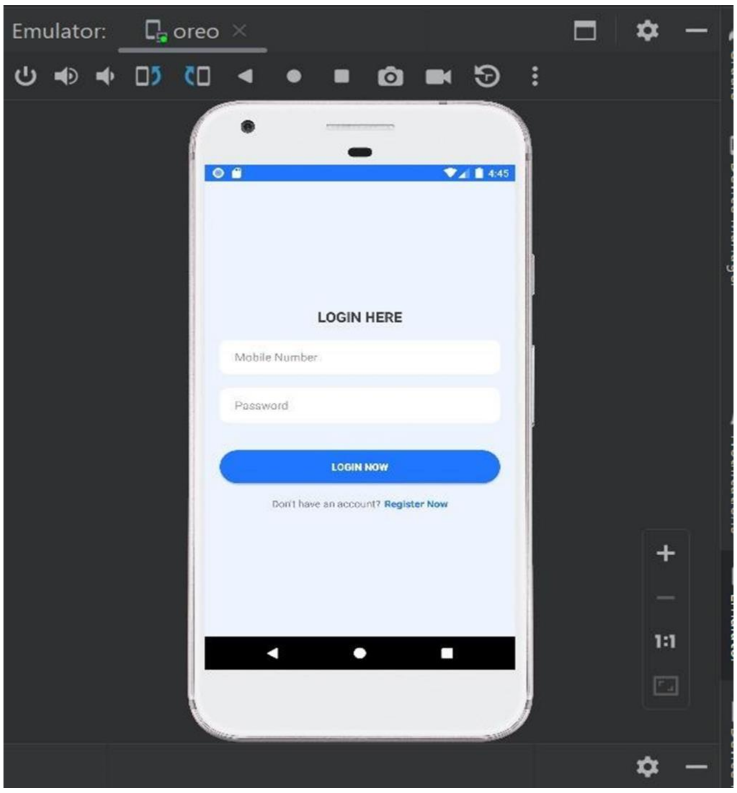
Review-The research paper aimed to investigate the effects of social media, specifically mobile apps like Snapchat and Instagram, on English speaking and reading skills. The paper focused on user attitudes, experiences, and perceptions towards the use of these social media platforms for learning English in a classroom setting. The research was conducted using various databases including Academic Search Complete, Education Source, ERIC, Library, Information Science, and Technology Abstracts, and Professional Development Collection. The findings revealed that learners with social media accounts, particularly Snapchat, showed less enthusiasm towards learning English, including recreational reading, compared to learners without social media accounts who had a more positive attitude towards learning English speaking skills. The research concluded that Instagram was a more effective social media platform for engaging and learning the English language, as it provided practical knowledge and promoted interactions, contributing to a greater understanding of the English language and its culture. However, further research is needed to explore the impact of Snapchat on learning English speaking skills.

## 5- FLOWCHART

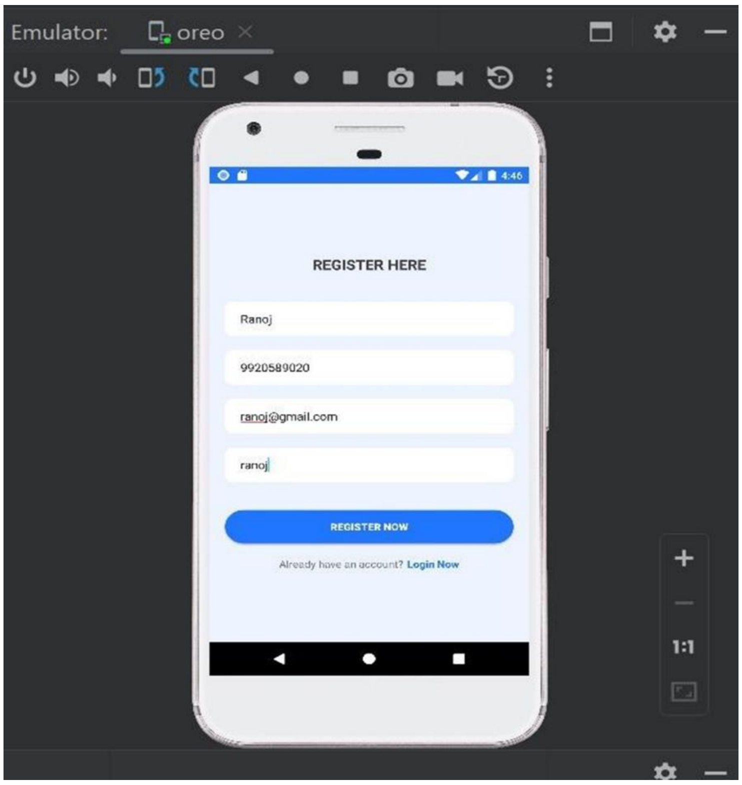

start
login/registe
authenticate information from firebase database
login
register
Retieve contacts information from Firestore cloud database
Home Screen
select contact
save the message in firebase as a seperate collection item
send a message
logout
end

## 6- RESULT

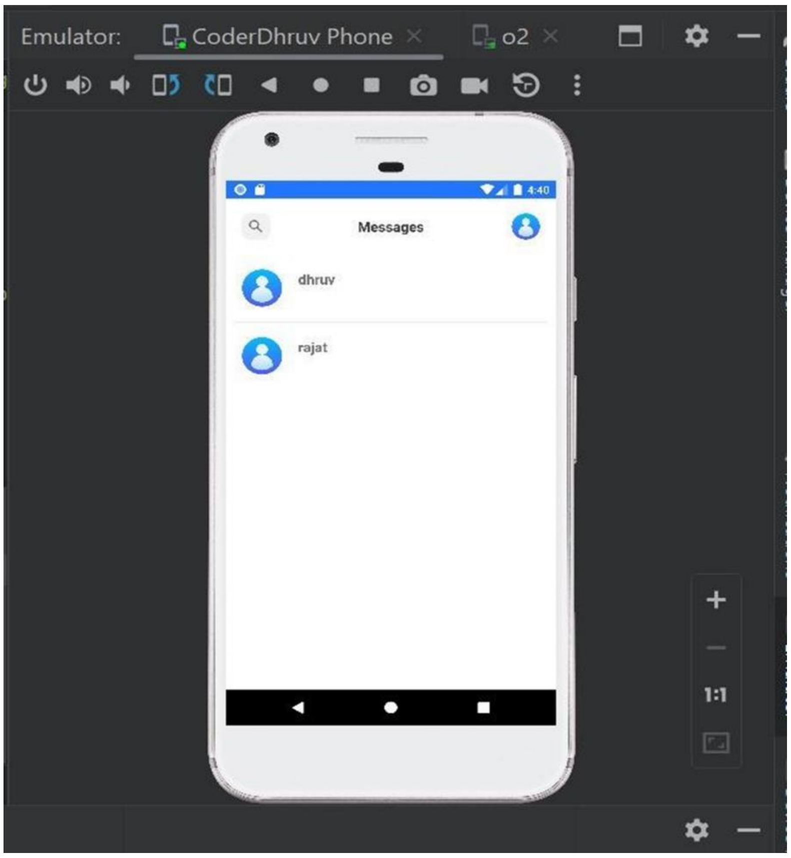
### Login page:



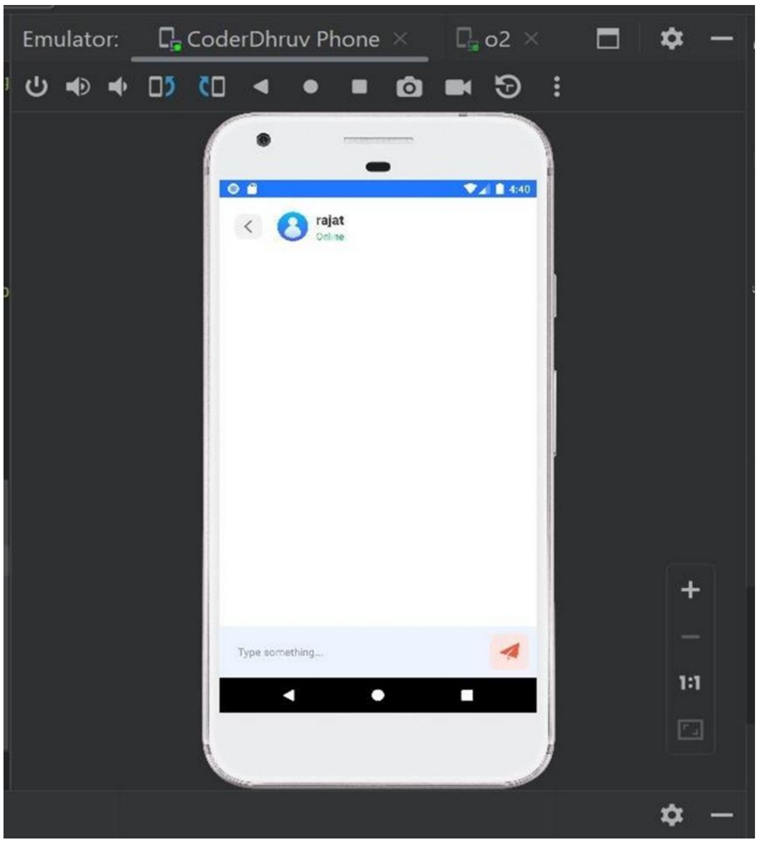
### Register Page:

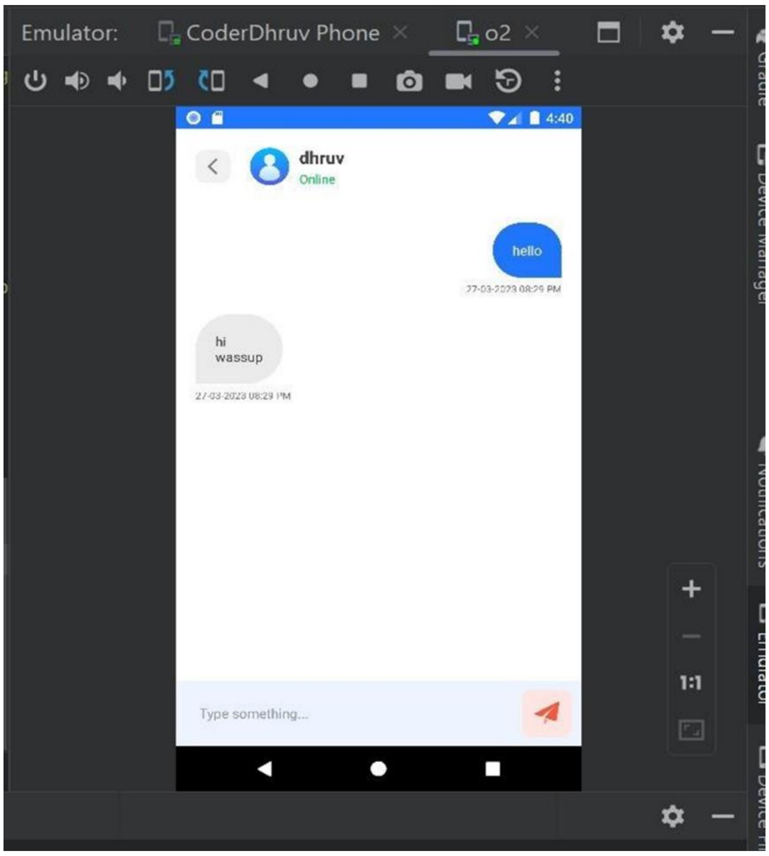


## Home Page:



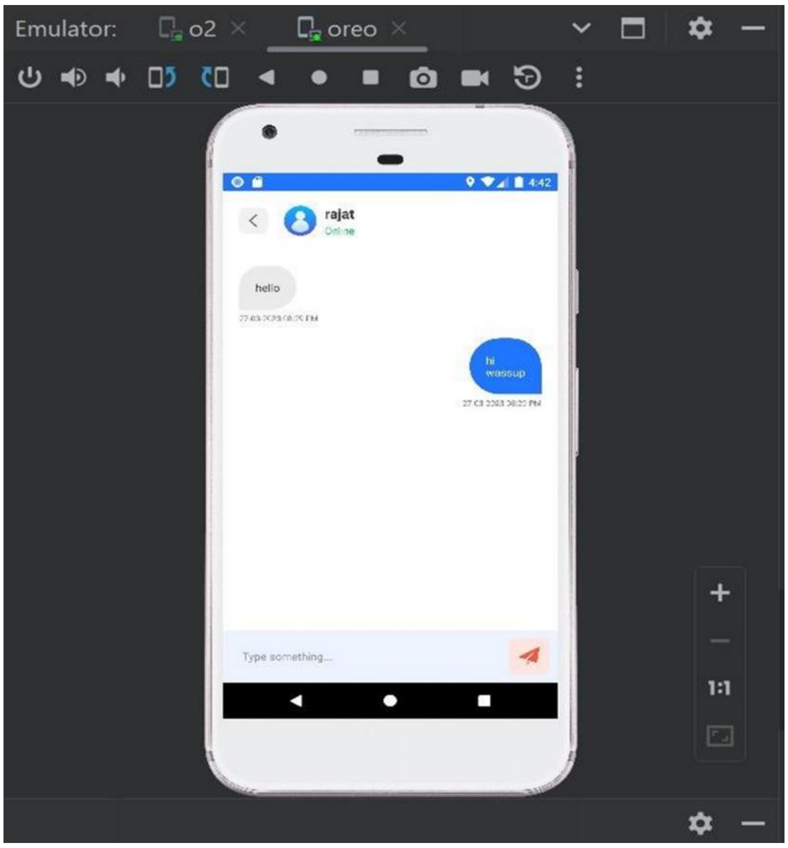
## Chat Screen:





hello
27-03-2023 08:29 PM

hi
wassup
27-03-2023 08:29 PM



hello
27-03-2023 08:29 PM

hi
wassup
27-03-2023 08:29 PM

# 7- ADVANTAGES & DISADVANTAGES

## 7.1 Advantages:

1. Real-time Communication: The chat application enables instant messaging, allowing users to engage in real-time conversations, fostering quick and efficient communication.
2. Global Connectivity: The application transcends geographical boundaries, allowing users to connect with individuals from around the world, expanding social and professional networks.
3. Accessibility: The chat application is accessible on multiple devices, including smartphones, tablets, and computers, providing flexibility and convenience for users.

## 7.2 Disadvantages:

1. Dependency on Internet Connectivity: The chat application relies on an internet connection, making it inaccessible in areas with poor or no network coverage.
2. Technical Issues: The chat application may encounter technical issues such as server outages, software bugs, or compatibility problems with certain devices or operating systems. Timely updates and maintenance are necessary to address these issues.
3. Privacy Concerns: While the chat application implements security measures, there may still be concerns about data privacy and the storage and handling of user information. Transparent privacy policies and adherence to data protection regulations are vital to address these concerns.
4. Reliance on Third-Party Services: The integration of third-party services, such as cloud storage or authentication providers, introduces a dependency on their availability, reliability, and compatibility with the chat application.
5. Multimedia Sharing: Users can share various forms of media, including photos, videos, and audio files, enhancing the richness and expressiveness of communication.

6. Group Collaboration: The chat application facilitates group chats, enabling teams or communities to collaborate, share ideas, and work together effectively.
7. Personalization: Users can customize their chat experience, including profile settings, notification preferences, and chat themes, tailoring the application to their individual preferences.

## 8- APPLICATIONS

The chat application holds immense potential and can be applied in various contexts, including:

**8.1 Personal Communication:** The chat application serves as a convenient and efficient tool for personal communication. Users can connect with family and friends, engage in one-on-one or group conversations, and share moments through text messages, photos, videos, and audio files. It enables individuals to stay connected regardless of geographical distances, fostering stronger relationships and facilitating easy communication.

**8.2 Business Collaboration:** In the business realm, the chat application becomes a valuable tool for collaboration and communication within teams and organizations. It allows team members to exchange messages, share files, and coordinate tasks in real-time. With the ability to create group chats and integrate with productivity tools, the application enhances productivity, streamlines workflows, and encourages efficient teamwork.

**8.3 Educational Institutions:** Educational institutions can utilize the chat application to facilitate communication among students, teachers, and staff members. It offers a platform for sharing course materials, discussing assignments, and organizing study groups. The application can also support announcements, notifications, and facilitate quick communication between educators and students, promoting effective learning and engagement.

**8.4 Community Engagement:** The chat application can serve as a hub for community engagement, connecting individuals with shared interests or goals. It enables the creation of communities, where users can participate in discussions, organize events, and share resources. This fosters a sense of belonging, promotes knowledge sharing, and cultivates meaningful connections within the community.

**8.5 Customer Support:** Businesses can leverage the chat application to provide efficient and personalized customer support. It allows customers to engage in real-time conversations with support representatives, addressing queries, resolving issues, and providing assistance. The application's multimedia sharing capabilities enable the exchange of relevant files or screenshots, facilitating effective troubleshooting and problem-solving.

**8.6 Professional Networking:** The chat application can be utilized as a platform for professional networking and building connections within industries or specific domains. Users can connect with like-minded professionals, engage in industry-related discussions, and exchange insights and opportunities. This enhances professional development, fosters collaboration, and opens doors for career advancement.

In summary, the chat application has diverse applications across personal, professional, educational, and community contexts. Its versatility and user-centric features make it a valuable tool for enhancing communication, collaboration, and engagement in various spheres of life.

# 9- CONCLUSION

In conclusion, the development of a chat application in its prototype phase is an exciting endeavor that holds great potential for future success. Through the iterative process of design, testing, and refinement, significant progress has been made in creating a functional and user-friendly chat application. However, as with any prototype, there are still areas that require further refinement and improvement.

The chat application's prototype phase has provided valuable insights and feedback from users, which have been invaluable in guiding future development. The feedback loop has allowed for continuous improvements, addressing usability issues, and refining the features based on user needs. The iterative nature of the prototype phase has also provided an opportunity to identify and fix bugs, ensuring a more stable and reliable product in the future.

Additionally, the prototype phase has paved the way for potential enhancements and expansion of the chat application. Future development can include integrating additional features such as encryption for enhanced security, integration with third-party platforms for seamless communication, and customization options to cater to specific user preferences.

In conclusion, the prototype phase of developing a chat application has been a crucial step in the product's evolution. While there are still areas to be refined and improved, the progress made so far has been promising, and the insights gained from user feedback and testing have been instrumental in shaping the application's future development. With continued efforts and dedication, the chat application has the potential to become a robust and widely used communication tool in the future.

## 10- FUTURE SCOPE

Based on the progress and insights gained during the prototype phase, there are several areas of future development and enhancement for the chat application:

1. Enhanced Security: Implementing end-to-end encryption for chat messages to ensure maximum privacy and security for users. This can include encryption of both text messages and media content shared within the application.
2. Integration with Additional Platforms: Expanding the chat application's compatibility by integrating it with other messaging platforms or social media networks. This would allow users to communicate seamlessly across different platforms, extending the reach and convenience of the application.
3. Customization Options: Providing users with more customization options to personalize their chat experience. This can include themes, fonts, chat bubble styles, and notification preferences, allowing users to tailor the application to their individual preferences.
4. Advanced Features: Introducing advanced features such as voice and video calling capabilities within the chat application. This would enable users to have richer and more immersive communication experiences, further enhancing the application's functionality.
5. Group Chat Management: Improving the management and administration of group chats by adding features such as the ability to assign administrators, control access and permissions, and create subgroups within larger chat groups.

6. Performance Optimization: Continuously optimizing the application's performance, scalability, and responsiveness to ensure smooth and efficient communication even with a growing user base.
7. Integration of AI and Chatbots: Exploring the integration of artificial intelligence (AI) technologies and chatbots to automate certain tasks, provide intelligent suggestions, and enhance user engagement within the chat application.
8. Localization and Internationalization: Adapting the chat application to support multiple languages, cultural nuances, and regional preferences, enabling users from diverse backgrounds to use the application comfortably.
9. Integration with Cloud Storage: Integrating cloud storage services to enable seamless sharing of larger files and media content, reducing the limitations on file size and ensuring efficient data transfer.
10. User Feedback and Continuous Improvement: Continuously gathering user feedback, conducting user surveys, and analyzing user behavior to identify areas for improvement and refine the application based on user needs and preferences.

By focusing on these future developments, the chat application can evolve into a comprehensive and feature-rich communication tool that meets the diverse needs of users and stays competitive in the dynamic messaging app market.

# 11- BIBILOGRAPHY

1- Tomás Dodds (2019) Reporting with WhatsApp: Mobile Chat Applications' Impact on Journalistic
Practices, Digital Journalism, 7:6, 725-745, DOI: 10.1080/21670811.2019.1592693

2- Dimas Natanael, Faisal, Dewi Suryani, Text Encryption in Android Chat Applications using Elliptical CurveCryptography (ECC), Procedia Computer Science,Volume 135,2018,Pages 283-291,ISSN 1877-0509,https://doi.org/10.1016/j.procs.2018.08.176

3- M. B. Kılıç , "Encryption Methods and Comparison of Popular Chat Applications", Advances in Artificial
Intelligence Research, vol. 1, no. 2, pp. 52-59, Sep. 2021

4- Agur, C. (2019). Insularized Connectedness: Mobile Chat Applications and News Production. Media andCommunication, 7(1), 179-188. doi:https://doi.org/10.17645/mac.v7i1.1802

5- S. Shukla, S. C. Gupta and P. Mishra, "Android-Based Chat Application Using Firebase," 2021 InternationalConference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2021, pp. 1-4, doi: 10.1109/ICCCI50826.2021.9402510.

6- Filbert, James (2010) “Developing a Multi-Purpose Chat Application for Mobile Distributed Systems onAndroid Platform”, Metropolia Ammattikorkeakoulu 2010

7- Mohamad Afendee, Mohamed and Abdullah, Muhammed and Mustafa, Man (2015) A secure chat application based on pure peer-to-peer architecture. Journal of Computer Science, 11 (5). pp. 723-729. ISSN15493636 [P]

8- Fei, Jennifer and Wolff, Jessica and Hotard, Michael and Ingham, Hannah and Khanna, Saurabh and Lawrence, Duncan and Tesfaye, Beza and Weinstein, Jeremy M. and Yasenov, Vasil and Hainmueller, Jens, Automated Chat Application Surveys Using Whatsapp: Evidence from Panel Surveys and a Mode Experiment.IZA Discussion Paper No. 15263, Available at SSRN: https://ssrn.com/abstract=4114839 or http://dx.doi.org/10.2139/ssrn.4114839

9- Falguni Patel, NirmalKumar Patel, Santosh Kumar Bharti “Lie-Sensor: A Live Emotion Verifier or a Licensor forChat Applications using Emotional Intelligence” arXiv:2102.11318

10- Abdulaziz Al Fadda, Hind, Determining How Social Media Affects Learning English: An Investigation of MobileApplications Instagram and Snap Chat in TESOL Classroom (2020). Arab World English Journal (AWEJ) Volume 11. Number1 March 2020 , Available at SSRN: https://ssrn.com/abstract=3581296 or http://dx.doi.org/10.2139/ssrn.3581296

# 12- APPENDIX

## GitHub link:

https://github.com/smartinternz02/SI-GuidedProject-525374-1688112449

## Drivelink of video demo:

https://drive.google.com/drive/folders/11Xyaun7qJvCktJsPjQz3rQCRkneNk51C?usp=sharing

## MainActivity:

```
class MainActivity : AppCompatActivity() {
    private lateinit var appBarConfiguration: AppBarConfiguration
    lateinit var binding: ActivityNavBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityNavBinding.inflate(layoutInflater)
        setContentView(binding.root)
        setSupportActionBar(binding.appBarNav.toolbar)
        val drawerLayout: DrawerLayout = binding.drawerLayout
        val navView: NavigationView = binding.navView
        val navController = findNavController(R.id.nav_host_fragment_content_nav)
        // Passing each menu ID as a set of Ids because each
        // menu should be considered as top level destinations.
        appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.nav_latest_messages
            ), drawerLayout
        )
        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }
    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        // Inflate the menu; this adds items to the action bar if it is present.
        menuInflater.inflate(R.menu.nav_options, menu)
        return true
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.sign_out -> {
                FirebaseDatabase.getInstance().getReference("/online-
users/${FirebaseAuth.getInstance().uid}").setValue(false)
                FirebaseAuth.getInstance().signOut()
                val intent = Intent(this, LauncherActivity::class.java)
                intent.flags = (Intent.FLAG_ACTIVITY_CLEAR_TASK) or (Intent.FLAG_ACTIVITY_NEW_TASK)
                startActivity(intent)
                finish()
            }
        }
        return super.onOptionsItemSelected(item)
    }
    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.nav_host_fragment_content_nav)
        return navController.navigateUp(appBarConfiguration) || super.onSupportNavigateUp()
    }
}
```

## ChatFragment.kt:

```
class ChatFragment : Fragment() {
    private lateinit var chatViewModel: ChatViewModel
    private lateinit var userViewModel: UserViewModel
    private lateinit var blocklistViewModel: BlocklistViewModel
    private var presentDialog = mutableStateOf(false)
    private var sendMessageState = mutableStateOf(TextFieldValue())
    override fun onCreateView(
            inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
        return ComposeView(requireContext()).apply {
```

```
        setContent {
            Box(alignment = Alignment.Center) {
                Chat()
                if (presentDialog.value) {
                    UserDialog()
                }
            }
        }
    }
}
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    chatViewModel = ViewModelProvider(requireActivity()).get(ChatViewModel::class.java)
    userViewModel = ViewModelProvider(requireActivity()).get(UserViewModel::class.java)
    blocklistViewModel = ViewModelProvider(requireActivity()).get(BlocklistViewModel::class.java)
    with (chatViewModel) {
        messages.value = listOf()
        if (tempUser != null) {
            listenForMessages(userViewModel.user.value?.uid!!, chatViewModel.tempUser!!.uid)
            listenForTypingIndicator(userViewModel.user.value?.uid!!, chatViewModel.tempUser!!.uid)
        }
        tempWriting.observe(viewLifecycleOwner, {
            if (it != "") {
                sendMessageState = mutableStateOf(TextFieldValue(it))
            }
        })
    }
}
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == 0 && resultCode == Activity.RESULT_OK && data != null) {
        chatViewModel.photoAttachmentUri = data.data
    }
    if (requestCode == 1 && resultCode == Activity.RESULT_OK && data != null) {
        chatViewModel.fileAttachmentUri = data.data
    }
}
var previousMessage: String? = null
var compareTime: Long = 0
@Composable
fun Chat() {
    Column {
        Box(alignment = Alignment.BottomCenter) {
            val scrollState = rememberScrollState()
            ScrollableColumn(
                    modifier = Modifier.fillMaxSize(),
                    scrollState = scrollState,
                    reverseScrollDirection = true
            ) {
                val messages by chatViewModel.messages.observeAsState()
                if (messages != null) {
                    messages?.forEach {
                        var consecutive = false
                        if (it.fromId == previousMessage) {
                            consecutive = true
                        }
                        var showTime = false
                        if ((it.time - compareTime) > 300) {
                            showTime = true
                        }
                        if (it.fromId == userViewModel.user.value?.uid) {
                            var modifier = if (chatViewModel.messages.value?.last() == it) {
                                Modifier.padding(bottom = 70.dp)
                            } else {
                                Modifier.padding(0.dp)
                            }
                            if (consecutive && !showTime) {
                                modifier = modifier.then(Modifier.padding(end = 42.dp))
                            }
                            ChatMessageFromItem(it, modifier, showTime, consecutive)
                        } else {
                            var modifier = if (chatViewModel.messages.value?.last() == it) {
                                Modifier.padding(bottom = 70.dp)
                            } else {
                                Modifier.padding(0.dp)
                            }
                            if (consecutive && !showTime) {
                                modifier = modifier.then(Modifier.padding(start = 42.dp))
                            }
                            ChatMessageToItem(it, modifier, showTime, consecutive)
                        }
                        previousMessage = it.fromId
                        compareTime = it.time
                    }
                    scrollState.scrollTo(0f)
                }
            }
            ChatSendMessageBar()
        }
    }
}
@Composable
fun ChatSendMessageBar() {
    val otherUsingTyping by chatViewModel.otherUserTyping.observeAsState()
    var modifier = Modifier.height(50.dp)
    if (otherUsingTyping != null && otherUsingTyping!!) {
        modifier = Modifier.height(68.dp)
    }
    Column(
            modifier = modifier
            .then(
                Modifier.background(
                    VerticalGradient(
                        listOf(Color.White, Color.Transparent),
                        startY = 100f,
                        endY = 0f
                    )
                )
            )
            .then(Modifier.background(Color(resources.getColor(R.color.half_white, null))))
    ) {
      if (otherUsingTyping != null && otherUsingTyping!!) {
        Text(
            text = "${chatViewModel.tempUser!!.username} is typing...",
            fontSize = 14.sp,
            fontStyle = FontStyle.Italic,
            modifier = Modifier.padding(start = 8.dp),
            fontWeight = FontWeight.Bold
        )
      }
      Row(
          verticalAlignment = Alignment.Bottom,
          modifier = Modifier.fillMaxHeight()
              .then(Modifier.padding(bottom = 4.dp))
      ) {
        Image(
            asset = vectorResource(id = R.drawable.ic_baseline_photo_camera_green_24),
            modifier = Modifier.size(35.dp)
                .then(Modifier.padding(start = 8.dp, bottom = 7.dp))
                .then(Modifier.clickable(onClick = {
                  if (chatViewModel.photoAttachmentUri != null) {
                    Toast.makeText(requireContext(), "Image already attached.",
Toast.LENGTH_LONG).show()
```

```
                        return@clickable
                    }
                    val intent = Intent(Intent.ACTION_PICK)
                    intent.type = "image/*"
                    startActivityForResult(intent, 0)
                }))
        )

        Image(
            asset = vectorResource(R.drawable.ic_baseline_folder_open_green_24),
            modifier = Modifier.size(42.dp)
                .then(Modifier.padding(start = 8.dp, end = 8.dp))
                .then(Modifier.clickable(onClick = {
                    if (chatViewModel.fileAttachmentUri != null) {
                        Toast.makeText(requireContext(), "File already attached.",
Toast.LENGTH_LONG).show()
                        return@clickable
                    }
                    val intent = Intent(Intent.ACTION_PICK)
                    intent.type = "*/*"
                    startActivityForResult(intent, 1)
                }))
        )
        sendMessageState = remember {
mutableStateOf(TextFieldValue(chatViewModel.tempWriting.value!!)) }

        val sendMessageModifier = Modifier.border(
            width = 1.5.dp,
            color = if (sendMessageState.value.text != "") Color(
                resources.getColor(
                    R.color.default_green,
                    null
                )
            ) else Color.Gray,
            shape = RoundedCornerShape(20)
        )
            .then(Modifier.padding(6.dp))
            .then(Modifier.height(20.dp))
            .then(Modifier.fillMaxWidth())
        Box(
            modifier = Modifier.padding(end = 5.dp, bottom = 5.dp, top = 5.dp)
                .then(Modifier.weight(1f)),
            alignment = Alignment.CenterStart
        ) {
          BasicTextField(
              value = sendMessageState.value,
              onValueChange = {
                sendMessageState.value = it
                chatViewModel.tempWriting.value = it.text
              },
              modifier = sendMessageModifier,
              cursorColor = Color.Black,
              textStyle = TextStyle(color = Color.Black)
          )
          Text(
              "Write something here...",
              modifier = Modifier.padding(start = 6.dp)
                    .then(Modifier.drawOpacity(if (sendMessageState.value.text == "") 0.7f else 0f)),
              color = (Color.Gray)
          )
        }
        TextButton(onClick = {
          with(chatViewModel) {
            when {
              photoAttachmentUri != null -> {
                uploadImage(userViewModel.user.value!!.username)
              }
```

```
                    fileAttachmentUri != null -> {
                        uploadFile(userViewModel.user.value!!.username)
                    }
                    else -> {
                        performSendMessage(userViewModel.user.value!!.username)
                    }
                }
            }
            sendMessageState.value = TextFieldValue()
        },
            modifier = Modifier.offset(x = (-4).dp, y = (-4).dp),
            contentPadding = PaddingValues(0.dp),
            colors = ButtonConstants.defaultButtonColors(
                backgroundColor = Color.Transparent,
                contentColor = Color(resources.getColor(R.color.default_green, null))
            )
        ) {
            Text("Send")
        }
    }
  }
}
@Composable
fun ChatMessageFromItem(message: ChatMessage, modifier: Modifier, showTime: Boolean, consecutive: 
Boolean = false) {
    if (showTime) {
        Column(horizontalAlignment = Alignment.CenterHorizontally, modifier = Modifier.fillMaxWidth()) {
            Text(message.timestamp)
        }
    }
    Column {
        Row(modifier = modifier) {
            Spacer(modifier = Modifier.weight(1f))
        Column(
            horizontalAlignment = Alignment.Start,
            modifier = Modifier.padding(if (consecutive && !showTime) 2.dp else 4.dp)
                .then(Modifier.background(Color(resources.getColor(R.color.default_green, null)),
RoundedCornerShape(8.dp)))
                .then(Modifier.padding(5.dp))
                .then(Modifier.preferredWidthIn(max = 250.dp))
        ) {
          if (message.text.isNotEmpty()) {
            Row {
              Text(message.text)
            }
          }
          with (message) {
            if (imageUrl != null) {
              ChatImage(imageUrl!!)
            }
            if (fileUrl != null && fileSize != null && fileType != null) {
              val file = FileAttachment(fileType!!, fileSize!!, fileUrl!!)
              ChatFile(file)
            }
          }
        }
        if (userViewModel.user.value?.profileImageUrl != null && (!consecutive || showTime)) {
          Column {
            PicassoImage(
                data = userViewModel.user.value?.profileImageUrl!!,
                modifier = Modifier.padding(top = 5.dp, end = 5.dp)
                    .then(Modifier.border(1.5.dp, Color(resources.getColor(R.color.default_green, null)),
CircleShape))
                    .then(Modifier.drawShadow(4.dp, CircleShape))
                    .then(
                        Modifier.size(35.dp)
                            .clip(CircleShape)
                    ),
                contentScale = ContentScale.Crop
            )
          }
        }
      }
    }
  }
  @Composable
  fun ChatMessageToItem(message: ChatMessage, modifier: Modifier, showTime: Boolean, consecutive:
Boolean = false) {
    if (showTime) {
      Column(horizontalAlignment = Alignment.CenterHorizontally, modifier = Modifier.fillMaxWidth()) {
        Text(message.timestamp)
      }
    }
    Column {
      Row(modifier = modifier) {
        if (chatViewModel.tempUser != null && (!consecutive || showTime)) {
          Column {
            val color = if (chatViewModel.onlineUsers.value!!.contains(message.fromId))
Color(resources.getColor(R.color.default_green, null)) else Color.Black
            PicassoImage(
                data = chatViewModel.tempUser!!.profileImageUrl,
                modifier = Modifier.padding(top = 5.dp, start = 5.dp)
                    .then(Modifier.border(1.5.dp, color, CircleShape))
                    .then(Modifier.drawShadow(4.dp, CircleShape))
                    .then(
                        Modifier.size(35.dp)
                            .clip(CircleShape)
                    )
                    .then(Modifier.clickable(onClick = {
                      presentDialog.value = true
                    })),
                    contentScale = ContentScale.Crop
                )
            }
        }
        Column(
            horizontalAlignment = Alignment.Start,
            modifier = Modifier.padding(if (consecutive && !showTime) 2.dp else 4.dp)
                .then(Modifier.background(Color.LightGray, RoundedCornerShape(8.dp)))
                .then(Modifier.padding(5.dp))
                .then(Modifier.preferredWidthIn(max = 250.dp))
        ) {
            if (message.text.isNotEmpty()) {
                Row {
                    Text(message.text)
                }
            }
            if (message.imageUrl != null) {
                ChatImage(message.imageUrl!!)
            }
        }
    }
  }
}
@Composable
fun ChatImage(imageUrl: String) {
  PicassoImage(
        data = imageUrl,
        modifier = Modifier.padding(top = 4.dp)
            .then(Modifier.fillMaxSize())
            .then(Modifier.clip(RoundedCornerShape(8.dp)))
            .then(Modifier.clickable(onClick = {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(imageUrl))
                startActivity(intent)
            })),
        contentScale = ContentScale.FillWidth
    )
}
@Composable
fun ChatFile(file: FileAttachment) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.padding(top = 4.dp)
            .then(Modifier.border(1.dp, Color.LightGray, RoundedCornerShape(8.dp)))
            .then(Modifier.background(Color.White, RoundedCornerShape(8.dp)))
            .then(Modifier.padding(5.dp))
            .then(Modifier.preferredWidthIn(max = 250.dp))
    ) {
        Text(file.fileType, fontWeight = FontWeight.Bold)
        Image(
            asset = vectorResource(id = R.drawable.ic_baseline_insert_drive_file_24),
            modifier = Modifier.size(50.dp)
        )
        if (file.fileSize > 1000) {
            Text("${file.fileSize.div(1000)}mB")
        } else {
            Text("${file.fileSize}kB")
        }
    }
}
@Composable
fun UserDialog() {
    Column(
        modifier = Modifier.fillMaxWidth(0.7f)
            .then(Modifier.background(Color.White, RoundedCornerShape(10.dp)))
            .then(Modifier.border(1.dp, Color.LightGray, RoundedCornerShape(10.dp)))
            .then(Modifier.padding(15.dp))
    ) {
      if (chatViewModel.tempUser != null) {
        val color = if (chatViewModel.onlineUsers.value!!.contains(chatViewModel.tempUser?.uid))
Color(resources.getColor(R.color.default_green, null)) else Color.Black
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.padding(bottom = 30.dp)
        ) {
          PicassoImage(
              data = chatViewModel.tempUser!!.profileImageUrl,
              modifier = Modifier.padding(end = 10.dp)
                  .then(Modifier.border(1.5.dp, color, CircleShape))
                  .then(Modifier.drawShadow(4.dp, CircleShape))
                  .then(
                      Modifier.size(50.dp)
                          .clip(CircleShape)
                  )
                  .then(Modifier.clickable(onClick = {
                    presentDialog.value = true
                  })),
              contentScale = ContentScale.Crop
          )
          Text(
              chatViewModel.tempUser!!.username,
              fontWeight = FontWeight.Bold,
              fontSize = 16.sp
          )
          Spacer(modifier = Modifier.weight(1f))
          TextButton(onClick = {
            presentDialog.value = false
          }) {
            Text("Close")
          }
```

```
            }
          }
          Row {
            Button(onClick = { /*TODO*/ }) {
              Text("View Profile")
            }
            Spacer(modifier = Modifier.weight(1f))
            Button(onClick = {
              if (chatViewModel.tempUser != null) {
                blocklistViewModel.addBlockedUser(chatViewModel.tempUser!!)
                presentDialog.value = false
                requireActivity().onBackPressed()
              }
            }) {
              Text("Block")
            }
          }
        }
      }
    }
```

## ChatViewModel.kt

```
class ChatViewModel: ViewModel() {
  var messages = MutableLiveData<List<ChatMessage>>()
  var onlineUsers = MutableLiveData<List<String>>()
  var tempWriting = MutableLiveData<String>()
  var otherUserTyping = MutableLiveData<Boolean>()
  var tempUser: User? = null
  var tempCid: String? = null
  var photoAttachmentUri: Uri? = null
  var fileAttachmentUri: Uri? = null
  var imageUrl: String? = null
  var fileAttachment: FileAttachment? = null
  init {
    messages.value = mutableListOf()
    onlineUsers.value = mutableListOf()
    tempWriting.value = ""
}
fun addMessage(message: ChatMessage) {
    val messagesCopy = messages.value?.toMutableList()
    messagesCopy?.add(message)
    messages.value = messagesCopy!!
}
fun listenForMessages(user: String, otherUser: String) {
    val ref = FirebaseDatabase.getInstance().getReference("/user-messages/$user/$otherUser/cid")
    ref.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            tempCid = snapshot.value.toString()
            val newRef = FirebaseDatabase.getInstance().getReference("/conversations/$tempCid")
            newRef.addChildEventListener(object : ChildEventListener {
                override fun onChildAdded(snapshot: DataSnapshot, previousChildName: String?) {
                    val message = snapshot.getValue(ChatMessage::class.java)
                    if (message != null) {
                        if (messages.value!!.isNotEmpty()) {
                            if (messages.value!!.last().time != message.time) {
                                addMessage(message)
                                return
                            } else {
                                return
                            }
                        }
                        addMessage(message)
                    }
                }
                override fun onChildChanged(snapshot: DataSnapshot, previousChildName: String?) {}
                override fun onChildRemoved(snapshot: DataSnapshot) {}
        override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?) {}


        override fun onCancelled(error: DatabaseError) {}
      })
    }
    override fun onCancelled(error: DatabaseError) {}
  })
}
fun listenForTypingIndicator(user: String, otherUser: String) {
  val ref = FirebaseDatabase.getInstance().getReference("/user-messages/$user/$otherUser/")
  ref.addValueEventListener(object : ValueEventListener {
    override fun onDataChange(snapshot: DataSnapshot) {
      val typing = snapshot.child("typing")
      if (typing.exists()) {
        otherUserTyping.value = typing.value == true
      }
    }
    override fun onCancelled(error: DatabaseError) {}
  })
}
private fun addOnlineUser(string: String) {
  val users = onlineUsers.value?.toMutableList()
  users?.add(string)
  onlineUsers.value = users!!
}
private fun removeOnlineUser(string: String) {
  val users = onlineUsers.value?.toMutableList()
  users?.remove(string)
  onlineUsers.value = users!!
}
fun checkOnlineUser(snapshot: DataSnapshot) {
  if (snapshot.value == true) {
    addOnlineUser(snapshot.key!!)
    } else if (snapshot.value == false && onlineUsers.value!!.contains(snapshot.key!!)) {
        removeOnlineUser(snapshot.key!!)
    }
}
fun listenForOnlineUsers() {
    val ref = FirebaseDatabase.getInstance().getReference("/online-users/")
    ref.addChildEventListener(object : ChildEventListener {
        override fun onChildAdded(snapshot: DataSnapshot, previousChildName: String?) {
            checkOnlineUser(snapshot)
        }
        override fun onChildChanged(snapshot: DataSnapshot, previousChildName: String?) {
            checkOnlineUser(snapshot)
        }
        override fun onChildRemoved(snapshot: DataSnapshot) {}
        override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?) {}
        override fun onCancelled(error: DatabaseError) {}
    })
}
fun uploadImage(from: String) {
    if (photoAttachmentUri == null) {
        return
    }
    val filename = UUID.randomUUID().toString()
    val ref = FirebaseStorage.getInstance().getReference("/images/$filename")
    ref.putFile(photoAttachmentUri!!)
            .addOnSuccessListener {
                ref.downloadUrl.addOnSuccessListener {
                    imageUrl = it.toString()
                    performSendMessage(from)
                }
            }
}
fun uploadFile(from: String) {
        if (fileAttachmentUri == null) { return }
        val filename = UUID.randomUUID().toString()
        val ref = FirebaseStorage.getInstance().getReference("/files/$filename")
        ref.putFile(fileAttachmentUri!!).addOnSuccessListener {
            ref.downloadUrl.addOnSuccessListener { it2 ->
                val attachedFileSize = "%.2f".format(it.metadata!!.sizeBytes.toDouble() / 1000).toDouble()
                fileAttachment = FileAttachment(it.metadata?.contentType!!, attachedFileSize, it2.toString())
                performSendMessage(from)
            }
        }
    }
    fun performSendMessage(from: String) {
        val text = tempWriting.value!!
        val fromId = FirebaseAuth.getInstance().uid ?: return
        val toId = tempUser!!.uid
        val ref = FirebaseDatabase.getInstance().getReference("/conversations/${tempCid}").push()
        val chatMessage: ChatMessage?
        val time = System.currentTimeMillis() / 1000
        val month = LocalDateTime.now().month.getDisplayName(java.time.format.TextStyle.FULL,
Locale.ENGLISH)
        val date = LocalDateTime.now().dayOfMonth
        val hour = LocalDateTime.now().hour
        val minute = LocalDateTime.now().minute
        val newHour = if (hour < 10) {
            "0$hour"
        } else {
            hour.toString()
        }
        val newMinute = if (minute < 10) {
            "0$minute"
        } else {
            minute.toString()
        }
```

```
//      val dateFormat = SimpleDateFormat("HH:mm", Locale.getDefault())
//      val timestring = dateFormat.format(messageData.timestamp)
    val timestamp = "$date $month, $newHour:$newMinute"
    if (text.isNotEmpty()) {
        chatMessage = ChatMessage(
            ref.key!!,
            text,
            fromId,
            toId,
            timestamp,
            time
        )
        if (imageUrl != null) {
            chatMessage.imageUrl = imageUrl
            imageUrl = null
            photoAttachmentUri = null
        }
        if (fileAttachment != null) {
            with (fileAttachment!!) {
                chatMessage.fileSize = fileSize
                chatMessage.fileType = fileType
                chatMessage.fileUrl = fileUrl
            }
            fileAttachment = null
            fileAttachmentUri = null
        }
        ref.setValue(chatMessage)
            .addOnSuccessListener {
                tempWriting.value = ""
            }
        val latestMessageRef =
            FirebaseDatabase.getInstance().getReference("/latest-messages/$fromId/$toId")
        latestMessageRef.setValue(chatMessage)
        val latestMessageToRef =
            FirebaseDatabase.getInstance().getReference("/latest-messages/$toId/$fromId")
        latestMessageToRef.setValue(chatMessage)
        val payload = buildNotificationPayload(chatMessage.text, from)
        apiService.sendNotification(payload)!!.enqueue(
            object : Callback<JsonObject?> {
              override fun onResponse(
                  call: Call<JsonObject?>?,
                  response: Response<JsonObject?>
              ) {
                if (response.isSuccessful) {
                  Log.d("TAG", "Notification sent.")
                }
              }
              override fun onFailure(
                  call: Call<JsonObject?>?,
                  t: Throwable?
              ) {}
            })
      }
    }
    private fun buildNotificationPayload(text: String, from: String): JsonObject {
      val payload = JsonObject()
      payload.addProperty("to", tempUser?.token)
      val data = JsonObject()
//      data.addProperty("title", FirebaseAuth.getInstance().uid)
//      data.addProperty("body", FirebaseManager.messageKey)
      data.addProperty("title", from)
      data.addProperty("body", text)
      payload.add("notification", data)
      Log.d("TAG", payload.toString())
      return payload
    }
    fun newConversation(user: User, userViewModel: UserViewModel, callback: () -> Unit) {
        fun moveOn() {
            tempUser = user
            callback()
        }
        val cid = UUID.randomUUID().toString()
        val ref = FirebaseDatabase.getInstance().getReference("/user-
messages/${userViewModel.user.value?.uid}/${user.uid}")
        val toRef = FirebaseDatabase.getInstance().getReference("/user-
messages/${user.uid}/${userViewModel.user.value?.uid}")
        ref.addListenerForSingleValueEvent(object: ValueEventListener {
            override fun onCancelled(p0: DatabaseError) {}
            override fun onDataChange(p0: DataSnapshot) {
                if (!p0.child("cid").exists()) {
                    ref.child("cid").setValue(cid)
                    toRef.child("cid").setValue(cid)
                            .addOnSuccessListener {
                                moveOn()
                            }
                } else {
                    moveOn()
                }
            }
        })
    }
}
```

## ContactsFragment.kt

```
class ContactsFragment : Fragment() {
    lateinit var contactsViewModel: ContactsViewModel
    override fun onCreateView(
            inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
```

```
    return ComposeView(requireContext()).apply {
        setContent {
            Contacts()
        }
    }
}
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    contactsViewModel = ViewModelProvider(requireActivity()).get(ContactsViewModel::class.java)
    contactsViewModel.fetchContacts()
}
@Composable
fun Contacts() {
    val contacts by contactsViewModel.contacts.observeAsState()
    if (contacts != null && contacts!!.isNotEmpty()) {
        ScrollableColumn {
            contacts?.forEach {
                ContactItem(it)
            }
        }
    } else {
        Column(horizontalAlignment = Alignment.CenterHorizontally, modifier = Modifier.fillMaxWidth()) {
            Text(
                    "You have no contacts!",
                    modifier = Modifier.padding(10.dp),
                    fontSize = 16.sp
            )
        }
    }
}
@Composable
fun ContactItem(user: User) {
    Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.height(75.dp)
                .then(Modifier.border(bottom = Border(0.5.dp, Color.LightGray)))
                .then(Modifier.fillMaxWidth())
        ) {
          PicassoImage(
              data = user.profileImageUrl,
              modifier = Modifier.padding(start = 10.dp)
                  .then(Modifier.background(Color.Black, CircleShape))
                  .then(Modifier.border(1.5.dp, Color.Black, CircleShape))
                  .then(
                      Modifier.preferredSize(60.dp)
                          .clip(CircleShape)
                  ),
              contentScale = ContentScale.Crop
          )
          Column(modifier = Modifier.padding(start = 15.dp)) {
            Text(
                user.username,
                modifier = Modifier.padding(bottom = 6.dp),
                fontWeight = FontWeight.Bold,
                fontSize = 16.sp
            )
          }
        }
    }
}
```

## ContactsViewModel.kt

```
class ContactsViewModel : ViewModel() {
  var contacts = MutableLiveData<List<User>>()
  init {
    contacts.value = mutableListOf()
  }
  fun addContact(user: User) {
    val mutableContacts = contacts.value?.toMutableList()
    mutableContacts?.add(user)
    contacts.value = mutableContacts!!
  }
  fun fetchContacts() {
    contacts.value = mutableListOf()
    val uid = FirebaseAuth.getInstance().uid
    FirebaseDatabase.getInstance().getReference("/users/$uid/contacts")
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onCancelled(p0: DatabaseError) {}
                override fun onDataChange(p0: DataSnapshot) {
                    p0.children.forEach {
                        val newRef = FirebaseDatabase.getInstance().getReference("/users/${it.value.toString()}")
                        newRef.addListenerForSingleValueEvent(object : ValueEventListener {
                            override fun onDataChange(snapshot: DataSnapshot) {
                                val user = snapshot.getValue(User::class.java)
                                if (user != null) {
                                    addContact(user)
                                }
                            }
                            override fun onCancelled(error: DatabaseError) {}
                        })
                    }
                }
            })
  }
}
```

## LoginFragment.kt

```
class LoginFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
): View {
   return ComposeView(context = requireContext()).apply {
      setContent {
         LoginForm()
      }
   }
}
@Composable
fun LoginForm() {
   Box(alignment = Alignment.TopEnd, modifier = Modifier.fillMaxSize()) {
      Box(
            alignment = Alignment.Center,
            modifier = Modifier.fillMaxSize()
                  .then(Modifier.background(Color(resources.getColor(R.color.default_green, null))))
      ) {
         ScrollableColumn(
               horizontalAlignment = Alignment.CenterHorizontally,
               modifier = Modifier.fillMaxHeight()
                     .then(Modifier.padding(top = 100.dp))
         ) {
            Image(
                  asset = imageResource(R.drawable.image_bird),
                  modifier = Modifier.height(125.dp)
            )
            Text(
                  "Log in to your account",
                  modifier = Modifier.padding(bottom = 25.dp, top = 10.dp),
                  fontSize = 26.sp,
                  fontWeight = FontWeight.Bold,
                  color = Color.White
            )
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
               val emailState = remember { mutableStateOf(TextFieldValue()) }
            val passwordState = remember { mutableStateOf(TextFieldValue()) }

            val usernameModifier = Modifier.border(width = 2.dp,

                color = if (emailState.value.text != "") Color.White else
Color(resources.getColor(R.color.half_white, null)),

                shape = RoundedCornerShape(10))

                .then(Modifier.padding(15.dp))

                .then(Modifier.preferredWidthIn(min = 300.dp))

            val passwordModifier = Modifier.border(width = 2.dp,

                color = if (passwordState.value.text != "") Color.White else
Color(resources.getColor(R.color.half_white, null)),

                shape = RoundedCornerShape(10))

                .then(Modifier.padding(15.dp))

                .then(Modifier.preferredWidthIn(min = 300.dp))

            Box(alignment = Alignment.CenterStart) {

              BasicTextField(

                  value = emailState.value,

                  onValueChange = { emailState.value = it },

                  maxLines = 1,

                  modifier = usernameModifier,

                  cursorColor = Color.White,

                  textStyle = TextStyle(color = Color.White)

              )

              Text(

                  "Email",

                  modifier = Modifier.padding(start = 14.dp).then(Modifier.drawOpacity(if
(emailState.value.text == "") 0.7f else 0f)),

                  color = (Color.White)

              )

            }

            Box(alignment = Alignment.CenterStart) {

              BasicTextField(

                  value = passwordState.value,

                  onValueChange = { passwordState.value = it },

                  maxLines = 1,
                keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
                visualTransformation = PasswordVisualTransformation(),
                modifier = Modifier.padding(top = 30.dp).then(passwordModifier),
                cursorColor = Color.White,
                textStyle = TextStyle(color = Color.White)
            )
            Text(
                "Password",
                modifier = Modifier.padding(start = 14.dp, top = 30.dp)
                    .then(Modifier.drawOpacity(if (passwordState.value.text == "") 0.7f else 0f)),
                color = (Color.White)
            )
        }
        Column(modifier = Modifier.align(Alignment.End)) {
            ClickableText(AnnotatedString("Forgot Password?"), onClick = {
            }, style = TextStyle(color = Color.White, fontWeight =
            FontWeight.Bold, fontSize = 16.sp), modifier = Modifier.padding(top = 12.dp, bottom =
15.dp))
        }
        Button(
            onClick = {
                if (emailState.value.text.isEmpty()) {
                    Toast.makeText(
                        requireContext(),
                        "Please enter an email address",
                        Toast.LENGTH_LONG
                    ).show()
                    return@Button
                }
                if (passwordState.value.text.isEmpty()) {
                    Toast.makeText(
                        requireContext(),
                        "Please enter a password",
                Toast.LENGTH_LONG
            ).show()
            return@Button
        }
        FirebaseAuth.getInstance().signInWithEmailAndPassword(
            emailState.value.text,
            passwordState.value.text
        )
            .addOnCompleteListener {
                if (it.isSuccessful) {
                    startActivity(Intent(requireContext(), MainActivity::class.java))
                    requireActivity().finish()
                }
            }
            .addOnFailureListener {
                Toast.makeText(
                    requireContext(),
                    "Login failed: ${it.message}",
                    Toast.LENGTH_LONG
                ).show()
            }
    },
    colors = ButtonConstants.defaultButtonColors(backgroundColor = Color.White),
    modifier = Modifier.padding(top = 10.dp)
        .then(Modifier.size(width = 335.dp, height = 44.dp))
) {
    Text("Sign In", color = Color(resources.getColor(R.color.default_green, null)))
}
}
}
}
Column {
```

```
            ClickableText(AnnotatedString("Register"), style = TextStyle(fontWeight = FontWeight.Bold, color = 
Color.White, fontSize = 16.sp),
                    modifier = Modifier.padding(end = 20.dp, top = 10.dp), onClick = {
            this@LoginFragment.findNavController().navigate(R.id.action_loginFragment_to_registerFragment)
            })
        }
      }
    }
}
```

## RegisterFragment.kt

```
class RegisterFragment : Fragment() {
    lateinit var registerViewModel: RegisterViewModel
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return ComposeView(context = requireContext()).apply {
            setContent {
                RegisterForm()
            }
        }
    }
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        registerViewModel = ViewModelProvider(requireActivity()).get(RegisterViewModel::class.java)
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == 0 && resultCode == Activity.RESULT_OK && data != null) {
            registerViewModel.profileImageUri.value = data.data
        }
    }
    @Composable
```

```
fun RegisterForm() {
  Box(alignment = Alignment.TopStart, modifier = Modifier.fillMaxSize()) {
    Box(
        alignment = Alignment.Center, modifier = Modifier.fillMaxSize()
        .then(Modifier.background(Color(resources.getColor(R.color.default_green, null))))
    ) {
      ScrollableColumn(
          horizontalAlignment = Alignment.CenterHorizontally,
          modifier = Modifier.fillMaxHeight()
              .then(Modifier.padding(top = 30.dp))
      ) {
        Image(
            asset = imageResource(R.drawable.image_bird),
            modifier = Modifier.height(125.dp)
        )
        Text(
            "Create a new account",
            modifier = Modifier.padding(bottom = 25.dp, top = 10.dp),
            fontSize = 26.sp,
            fontWeight = FontWeight.Bold,
            color = Color.White
        )
        Column(horizontalAlignment = Alignment.CenterHorizontally) {
          val emailState = remember { mutableStateOf(TextFieldValue()) }
          val usernameState = remember { mutableStateOf(TextFieldValue()) }
          val passwordState = remember { mutableStateOf(TextFieldValue()) }
          val passwordConfirmationState = remember { mutableStateOf(TextFieldValue()) }
          fun createModifier(state: MutableState<TextFieldValue>): Modifier {
            return Modifier.border(width = 2.dp,
                color = if (state.value.text != "") Color.White else
Color(resources.getColor(R.color.half_white, null)),
                shape = RoundedCornerShape(10))
                .then(Modifier.padding(15.dp))
                .then(Modifier.preferredWidthIn(min = 300.dp))
          }
          val emailModifier = createModifier(emailState)
          val usernameModifier = createModifier(usernameState)
          val passwordModifier = createModifier(passwordState)
          val passwordConfirmModifier = createModifier(passwordConfirmationState)
          TextButton(onClick = {
            val intent = Intent(Intent.ACTION_PICK)
            intent.type = "image/*"
            startActivityForResult(intent, 0)
          }) {
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
              val image by registerViewModel.profileImageUri.observeAsState()
              if (image != null) {
                PicassoImage(
                      data = image!!,
                      modifier = Modifier.padding(bottom = 10.dp)
                            .then(Modifier.drawShadow(20.dp, CircleShape))
                            .then(Modifier.size(150.dp).clip(CircleShape))
                            .then(Modifier.border(2.dp, Color.White, CircleShape)),
                      contentScale = ContentScale.Crop
                )
              } else {
                Image(
                      asset = vectorResource(R.drawable.ic_baseline_account_circle_white_24),
                      modifier = Modifier.size(100.dp, 100.dp)
                )
              }
              Text("Choose a profile picture", color = Color.White, fontWeight = FontWeight.Bold,
fontSize = 16.sp)
            }
          }
          Box(alignment = Alignment.CenterStart, modifier = Modifier.padding(top = 20.dp)) {
```

```
                    BasicTextField(
                        value = emailState.value,
                        onValueChange = { emailState.value = it },
                        maxLines = 1,
                        modifier = emailModifier,
                        cursorColor = Color.White,
                        textStyle = TextStyle(color = Color.White)
                    )
                    Text(
                        "Email",
                        modifier = Modifier.padding(start = 14.dp).then(Modifier.drawOpacity(if
(emailState.value.text == "") 0.7f else 0f)),
                        color = (Color.White)
                    )
                }
                Box(alignment = Alignment.CenterStart) {
                    BasicTextField(
                        value = usernameState.value,
                        onValueChange = { usernameState.value = it },
                        maxLines = 1,
                        modifier = Modifier.padding(top = 20.dp).then(usernameModifier),
                        cursorColor = Color.White,
                        textStyle = TextStyle(color = Color.White)
                    )
                    Text(
                        "Username",
                        modifier = Modifier.padding(start = 14.dp, top = 20.dp)
                            .then(Modifier.drawOpacity(if (usernameState.value.text == "") 0.7f else 0f)),
                        color = (Color.White)
                    )
                }
                Box(alignment = Alignment.CenterStart) {
                    BasicTextField(
                        value = passwordState.value,
                        onValueChange = { passwordState.value = it },
                        maxLines = 1,
                        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
                        visualTransformation = PasswordVisualTransformation(),
                        modifier = Modifier.padding(top = 20.dp).then(passwordModifier),
                        cursorColor = Color.White,
                        textStyle = TextStyle(color = Color.White)
                    )
                    Text(
                        "Password",
                        modifier = Modifier.padding(start = 14.dp, top = 20.dp)
                            .then(Modifier.drawOpacity(if (passwordState.value.text == "") 0.7f else 0f)),
                        color = (Color.White)
                    )
                }
                Box(alignment = Alignment.CenterStart) {
                    BasicTextField(
                        value = passwordConfirmationState.value,
                        onValueChange = { passwordConfirmationState.value = it },
                        maxLines = 1,
                        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
                        visualTransformation = PasswordVisualTransformation(),
                        modifier = Modifier.padding(top = 20.dp).then(passwordConfirmModifier),
                        cursorColor = Color.White,
                        textStyle = TextStyle(color = Color.White)
                    )
                    Text(
                        "Confirm Password",
                        modifier = Modifier.padding(start = 14.dp, top = 20.dp)
                            .then(Modifier.drawOpacity(if (passwordConfirmationState.value.text == "") 0.7f
else 0f)),
                        color = (Color.White)
                    )
                }
                Button(
                    onClick = {
                        val email = emailState.value.text
                        val username = usernameState.value.text
                        val password = passwordState.value.text
                        val passwordConfirm = passwordConfirmationState.value.text
                        if (email.isEmpty() ||
                            username.isEmpty() ||
                            password.isEmpty() ||
                            passwordConfirm.isEmpty()) {
                            Toast.makeText(requireContext(), "Please enter an email address, username and 
password.", Toast.LENGTH_LONG).show()
                            return@Button
                        }
                        if (password != passwordConfirm) {
                            Toast.makeText(requireContext(), "Passwords do not match", 
Toast.LENGTH_LONG).show()
                            return@Button
                        }
                        if (registerViewModel.profileImageUri.value != null) {
                            FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, password)
                                .addOnCompleteListener { task ->
                                    if (task.isSuccessful) {
                                        Log.d("Main", "User created with ID: ${task.result?.user?.uid}")

registerViewModel.uploadImageToFirebase(registerViewModel.profileImageUri.value!!) {
                                        val uid = FirebaseAuth.getInstance().uid ?: ""
                                        val user = User(
                                            uid,
                                            username,
                                            it,
                                            email,
                                        null
                                        )
                                        registerViewModel.saveUserToDatabase(user) {
                                        val intent = Intent(requireActivity(), MainActivity::class.java)
                                        intent.flags =
Intent.FLAG_ACTIVITY_CLEAR_TASK.or(Intent.FLAG_ACTIVITY_NEW_TASK)
                                        startActivity(intent)
                                        }
                                    }
                                }
                            }
                            .addOnFailureListener {
                                Toast.makeText(requireContext(), "Invalid parameters: ${it.message}",
Toast.LENGTH_LONG).show()
                            }
                    }
                },
                colors = ButtonConstants.defaultButtonColors(backgroundColor = Color.White),
                modifier = Modifier.padding(top = 30.dp, bottom = 30.dp).then(Modifier.size(width =
335.dp, height = 44.dp))
        ) {
                Text("Register", color = Color(resources.getColor(R.color.default_green, null)))
        }
      }
    }
  }
  Column {
    ClickableText(AnnotatedString("Back"), style = TextStyle(fontWeight = FontWeight.Bold, color =
Color.White, fontSize = 16.sp),
        modifier = Modifier.padding(start = 20.dp, top = 10.dp), onClick = {

this@RegisterFragment.findNavController().navigate(R.id.action_registerFragment_to_loginFragment)
    })
  }
}
```

```
  }
}
```

## RegisterViewModel.kt

```
class RegisterViewModel: ViewModel() {
  var profileImageUri = MutableLiveData<Uri>()
  fun uploadImageToFirebase(uri: Uri, callback: (String) -> Unit) {
    val filename = UUID.randomUUID().toString()
    val ref = FirebaseStorage.getInstance().getReference("/images/$filename")
    ref.putFile(uri)
      .addOnSuccessListener {
        Log.d("Main", "Successfully uploaded image: ${it.metadata?.path}")
        ref.downloadUrl.addOnSuccessListener { it2 ->
          Log.d("Main", "File Location: $it2")
          callback(it2.toString())
        }
      }
      .addOnFailureListener {
        Log.d("Main", "Image upload failed")
      }
  }
  fun saveUserToDatabase(user: User, callback: () -> Unit) {
    val uid = FirebaseAuth.getInstance().uid ?: ""
    val ref = FirebaseDatabase.getInstance().getReference("/users/$uid")
    val contactRef = FirebaseDatabase.getInstance().getReference("/users/$uid/contacts")
    ref.setValue(user)
      .addOnSuccessListener {
        callback()
      }
    contactRef.setValue(listOf<String>())
  }
}
```