# CHATCONNECT-A REAL-TIME CHAT AND COMMUNICATION APP

**PROJECT REPORT**

*Submitted by*

**ARUL S R (20203111506218)**

**ASPIN LIJO K S (20203111506219)**

**ASWIN J (20203111506220)**

**BANESH A (20203111506221)**

*Submitted to Manonmaniam Sundaranar University. Tirunelveli*

*In partial fulfilment for the award of*

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

*Under the Guidance of*

**Prof.D.H.KITTY SMAILIN M.Sc.,M.Phill**


**DEPARTMENT OF PG COMPUTER SCIENCE**

**NESAMONY MEMORIAL CHRISTIAN COLLEGE,**

**MARTHANDAM**

**KANYAKUMARI DISTRICT -629165**

*Re-accredited with 'A' grade by NAAC*

**MARCH 2023**

# DECLARATION

I hereby declare that the project work entitled"**CHATCONNECT-A REAL-TIME CHAT AND COMMUNICATION APP"** is an original work done by me in partial fulfillment of the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE.** The study has been carried under the guidance of **Prof.D.H.KITTY SMAILIN M.Sc.,M.Phil Department of PG COMPUTER SCIENCE, Nesamony Memorial Christian College, Marthandam.** I declare that this work has not been submitted elsewhere for the award of any degree.

Place:Marthandam.

Arul S R (20203111506218)
Aspin lijo K S (20203111506219 )
Aswin J (20203111506220)
Benish A (20203111506221)

# ACKNOWLEDGEMENT

First of all I offer my prayers to god almighty for blessing me to complete this internship programme successfully

I express my sincere thanks to **, Dr.K.Paul Raj M.Sc., M.Phil., M.Ed., M.Phil(Edu).,Ph.D.,** Principal Nesamony Memorial Christian college, for his official support to do my work.

I express my profound thanks to **Dr. Dr.D.Latha M.Sc., M.Phil., Ph.D.,** HOD, Department of PG Computer Science for his encouragement given to undertake this project.

I extend my deep sense of gratitude to **, Prof**. **D.H.KITTY SMAILIN M.Sc.,M.Phil** Department of of PG Computer Science for guiding me in completing this project work successfully.

My special thanks to other Faculty members of Department of PG Computer Science for Guiding me in Completing this project.

Then I want to thank for Naanmudhalvan smartinternz team for giving me as a wonderful opportunity for doing a android project and gave more knowledge about the android application by free course videos.

I also wish to extend a special word of thanks to my parents, friends and all unseen hands that helped me for completing this project work successfully.

# CONTENTS

# 8APPENDIX

# 1INTRODUCTION

Internet-based simultaneous and cooperation which is time based on text and multimedia has become major area of research. Applications for the same are currently not defined in a good manner. The term ”collaborative application” is currently used to refer to any software programme that allows users to connect to one another in order to connect information in writing or through video in real-time or very close to real-time. As a result, several programmes nowadays make the claim to be collaborative. Internet users who are online or those who are constantly using the internet can talk directly with one another using a feature or programme called a chat room. Users of chat apps can communicate even when they are far apart. To be accessed by many people, this feature must be problem solving time and platform independent. The chatapp is currently being created by a lot of programmers. Numerous chat applications have their own pros and cons.We examined the available messaging systems before beginning the development of the project. We already knew that there were a number of messaging services and chat programmes. We had never, however, examined their tools in-depth to see whether they were enough for developers. We quickly became aware that none of the locations were moving in our direction. Some of them lacked characteristics that we thought were essential, while others offered room for improvement. We looked into various platforms like Gitter, Slack, Whatsapp, Telegram, Messenger, Discord, Skype, Flowdock, etc. There are billions of users of the listed applications worldwide. These businesses rank among the strongest in the industry. They make more profit each year and hire a vast group of users to work on new features for their releases to keep up with other companies. These applications use a variety of features and procedures to protect the privacy of their users’ data. Today, the most common crime is data theft, which is committed by the majority of people. These days, a lot of instances involving the loss of personal data are being filed. Hence, entities need to protect data security against data breaches. The chatting system should also provide simultaneous operations like send and receive. Both transmitting and receiving are possible in this application. Based on the background research, the problem with real time chat applications is that different application have

different features. We are trying to bring all features like sending invitation, online indicator, notify on typing, storage of messages in database, chatting, audio and video call, screen sharing in one application. +?nm,./ Contrary to popular opinion, having a few applications available is a good thing. Based on their experience, we were able to gather insights for what to develop and how, and choose the technologies and techniques to implement. Checking their blogs was usually all that was required. Companies like Slack usually publish updates on their development. Several times, we had to look on the internet to learn about our alternatives and choose the one we felt was necessary.

Communication is a mean for people to exchange messages. Messaging apps (a.k.a. social messaging or chat applications) are apps and platforms that enable instant messaging. According to the survey the group of users prefer WhatsApp and like to communicate using Emoji. 51% of the group uses the chat applications on an average of 1-2 hours a day. Messaging apps now have more global users than traditional social network which means they will play an increasingly important role in the distribution of digital journalism in the future. Our project is related to a new way of chatting with people. Chatting and communicating with people through internet is becoming common to people and is connecting people all over the world. Mainly, chatting apps in today's world mainly focus on connecting people, providing users with more features like GIFs, stickers etc. But this app, is different from them. This chatting application includes chatting through internet using IP address. It mainly focuses on chatting and connects people all around the world. Mostly, chatting applications like WhatsApp requires mobile no. of the person and then we can chat and connect with the person. But here, the person only has to login with the system, and then he can connect with the people which he wants with. The Discuss Chat app is an open - source chatting app. It means people all over the world can join the chat between people easily. We can check and see the people joining and leaving the chat group. For using the app, firstly we have to register our name in the application. After registration, the person will be given a particular IP address, which is only used by that person, so that people will same name can be differentiated easily. The IP address can only be seen by the person which is registered under that name. Once, the registration of the person is done, he can join the chat room. The chatting between 2 people can be easily converted into group, as the people chatting easily know if there is another person, who wants to join the chat between them. If we have to chat with a specific person, then we just have to know the name of the person and its IP

address. It's different from the present chatting applications, as it includes the personal information of the person, which gets accessed by the person which is following him or is friends with the person. This can save the person from sharing his personal stuff to strangers, without his consent. One of the features of this application is that, if a new person connects with them while chatting in person or in group, then the app shows the people the person is connecting or joining them while being in the group. Not only group chatting, but personal chatting between people also takes place. Only during group chatting, people can enter the chat room. While personal chatting, the chat and talk between people is kept encrypted between them. In simple words, the chat between them cannot be read by other people, not even by the app. One of the important features of this app, is related to group chatting. If for example, there are 2 groups in which there are 15 – 20 people. If more than 5 people are common in both the groups, and an important message or file is shared in one of the groups, then the person which shared the information is asked whether the same information is to be shared in the other group. If yes, then the information will be sent to the other group directly, without any human interference

## 1.1 Overview

### Technical Overview

Discuss the technical requirements for developing a real-time chat app using Android Studio.Explain the key technologies and tools involved, such as Android SDK, Firebase or other backend services for real-time messaging and user authentication, and XML for UI design.Provide an overview of the architecture of a real-time chat app, including the client-side and server-side components, and the flow of data and messages between them.Discuss the challenges and considerations in developing a real-time chat app, such as handling real-time updates, managing user authentication and authorization, handling network connectivity, and ensuring data security.

Android Studio is the official Integrated Development Environment (IDE) for developing Android applications. It is a powerful tool that allows developers to build high-quality applications for the Android platform. In this article, we will guide you through the process of setting up Android Studio on your computer. The steps on how to use Android Studio.

### Step 1: Download Android Studio

To set up Android Studio, you need to first download the IDE from the official Android Studio download page. Choose the version that is compatible with your operating system, and download the installer. Android Studio is available for Windows, macOS, and Linux. Once the download is complete, run the installer and follow the instructions to install Android Studio on your computer.

**Step 2: Install the Required Components**

During the installation process, Android Studio will prompt you to install the required components. These include the Android SDK, Android Virtual Device (AVD) Manager, and the Android Emulator. The Android SDK is a collection of libraries and tools that developers use to build Android applications. The AVD Manager is used to create and manage virtual devices for testing applications. The Android Emulator is a virtual device that allows developers to test their applications without having to use a physical device.

**Step 3: Configure Android Studio**

After installing Android Studio, you need to configure it before you can start using it. When you launch Android Studio for the first time, you will be prompted to configure the IDE. Choose the "Standard" configuration and click on "Next". In the next screen, you can choose the theme of the IDE and click on "Next" again. You can also customize the settings based on your preferences.

**Step 4: Create a New Project**

Once Android Studio is configured, you can start creating your first Android application. To create a new project, click on "Start a new Android Studio project" on the welcome screen, or select "New Project" from the "File" menu. You will be prompted to choose the project name, package name, and other project details. You can also choose the minimum SDK version, which determines the minimum version of Android that the application can run on.

**Step 5: Build Your Application**

Once your project is created, you can start building your application using the various tools and features provided by Android Studio. You can use

the visual layout editor to design the user interface, write code in Java or Kotlin, and use the Android SDK to access device features such as the camera, sensors, and GPS. You can also use the built-in debugging tools to troubleshoot issues and optimize your application.

**Step 6: Test Your Application**

Testing your application is an important step in the development process. Android Studio comes with an emulator that allows you to test your application on different virtual devices. You can also connect your Android device to your computer and test your application directly on the device. Use the "Run" button in Android Studio to launch your application and test it on the emulator or device. You can also use the built-in profiler to analyze the performance of your application and identify any bottlenecks or performance issues. In conclusion, setting up Android Studio is a crucial step in developing Android applications. By following these steps, you can easily set up Android Studio on your computer and start building high-quality Android applications. Android Studio provides a powerful set of tools and features that make the development process easier and more efficient.

**Components in Android Studio**

**1. Manifest File**

The AndroidManifest.xml file is a crucial component of any Android application. It provides essential information about the application to the Android operating system, including the application's package name, version, permissions, activities, services, and receivers. The manifest file is required for the Android system to launch the application and to determine its functionality. Here are some of the key uses of the manifest file in an Android application:

Declaring Application Components: The manifest file is used to declare the various components of an Android application, such as activities, services, and broadcast receivers. These components define the behavior and functionality of the application, and the Android system uses the manifest file to identify and launch them.

Specifying Permissions: Android applications require specific permissions to access certain features of the device, such as the camera, GPS, or storage.

The manifest file is used to declare these permissions, which the Android system then checks when the application is installed. If the user has not been granted the required permissions, the application may not be able to function correctly.

Defining App Configuration Details: The manifest file can also be used to define various configuration details of the application, such as the application's name, icon, version code and name, and supported screens. These details help the Android system to identify and manage the application properly.

Declaring App-level Restrictions: The manifest file can be used to declare certain restrictions at the app level, such as preventing the application from being installed on certain devices or specifying the orientation of the app on different screens.

In summary, the manifest file is an essential part of any Android application. It provides important information about the application to the Android system and enables the system to launch and manage the application correctly. Without a properly configured manifest file, an Android application may not be able to function correctly, or it may not be installed at all.

**2. Build.gradle**

**Gradle**

build.gradle is a configuration file used in Android Studio to define the build settings for an Android project. It is written in the Groovy programming language and is used to configure the build process for the project. Here are some of the key uses of the build.gradle file:

Defining Dependencies: One of the most important uses of the build.gradle file is to define dependencies for the project. Dependencies are external libraries or modules that are required by the project to function properly. The build.gradle file is used to specify which dependencies the project requires, and it will automatically download and include those dependencies in the project when it is built.

Setting Build Options: The build.gradle file can also be used to configure various build options for the project, such as the version of the Android SDK to use, the target version of Android, and the signing configuration for the project.

Configuring Product Flavors: The build.gradle file can be used to configure product flavors for the project. Product flavors allow developers to create different versions of their application with different features or configurations. The build.gradle file is used to specify which product flavors should be built, and how they should be configured.

Customizing the Build Process: The build.gradle file can also be used to customize the build process for the project. Developers can use the build.gradle file to specify custom build tasks, define build types, or customize the build process in other ways.

Overall, the build.gradle file is a powerful tool for configuring the build process for an Android project. It allows developers to define dependencies, configure build options, customize the build process, and more. By understanding how to use the build.gradle file, developers can optimize the build process for their projects and ensure that their applications are built correctly and efficiently.

**3. Git**

Git is a popular version control system that allows developers to track changes to their code and collaborate with other team members. Android Studio includes built-in support for Git, making it easy to manage code changes and collaborate with others on a project. Here are some of the key uses of Git in Android Studio:

Version Control: Git allows developers to track changes to their code over time. This means that they can easily roll back to a previous version of their code if needed, or review the changes made by other team members.

**Collaboration**: Git enables multiple developers to work on the same codebase simultaneously. Developers can work on different features or parts of the codebase without interfering with each other, and merge their changes together when they are ready.

**Branching and Merging:** Git allows developers to create branches of their codebase, which can be used to work on new features or bug fixes without affecting the main codebase. When the changes are complete, the branch can be merged back into the main codebase.

**Code Review**: Git allows team members to review each other's code changes before they are merged into the main codebase. This can help ensure that the code is of high quality and meets the project's requirements.

Android Studio includes a built-in Git tool that allows developers to perform common Git tasks directly within the IDE. Developers can create new repositories, clone existing ones, and manage branches and commits. Android Studio also provides a visual diff tool that makes it easy to see the changes made to the codebase over time. To use Git in Android Studio, developers need to first initialize a Git repository for their project. Once the repository is set up, they can use the Git tool in Android Studio to manage changes to their code, collaborate with others, and review code changes.

In summary, Git is a powerful version control system that is essential for managing code changes and collaborating with other team members. Android Studio includes built-in support for Git, making it easy for developers to manage their code changes directly within the IDE.

**4. Debug**

Debugging is an essential part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. Here are some of the key uses of debugging in Android Studio.

**Identifying Issues:** Debugging helps developers identify issues in their code by allowing them to inspect variables, evaluate expressions, and step through the code line by line. This allows developers to pinpoint exactly where a problem is occurring and fix it more quickly.

**Optimizing Performance:** Debugging can also be used to optimize the performance of an application by identifying bottlenecks or areas of inefficient

code. By profiling an application while it is running, developers can identify areas of the code that are causing slow performance and make changes to improve performance.

**Testing and Validation:** Debugging is also useful for testing and validating an application. By stepping through code and inspecting variables, developers can ensure that the application is behaving as expected and that it is producing the desired output.

Android Studio provides a comprehensive set of debugging tools, including breakpoints, watches, and the ability to evaluate expressions in real time. Developers can use these tools to inspect variables, step through code, and identify issues in their applications.

To use the debugging tools in Android Studio, developers need to first configure their project for debugging by adding breakpoints to their code. Breakpoints are markers that tell the debugger to pause execution at a certain point in the code. Once the breakpoints are set, developers can run their application in debug mode and step through the code line by line, inspecting variables and evaluating expressions as they go.

In summary, debugging is a critical part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. By using these tools, developers can optimize performance, test and validate their code, and improve the quality of their applications.

**5. App Inspection**

**Inspector**

App Inspection is a feature in Android Studio that allows developers to inspect and debug their Android applications. It provides a suite of tools for analyzing the performance of the application, identifying and fixing errors, and optimizing the code. Here are some of the key features and uses of App Inspection:

**Performance Analysis:** App Inspection provides tools for analyzing the performance of an Android application. Developers can use these tools to

identify performance bottlenecks, such as slow database queries or inefficient network requests, and optimize the code to improve performance.

**Error Detection and Debugging:** App Inspection allows developers to detect and debug errors in their Android applications. It provides tools for tracking down errors and identifying the root cause of the issue, making it easier to fix bugs and improve the stability of the application.

**Memory Management:** App Inspection provides tools for managing the memory usage of an Android application. Developers can use these tools to identify memory leaks and optimize the code to reduce memory usage, which can improve the performance and stability of the application.

**Network Profiling:** App Inspection includes tools for profiling network traffic in an Android application. Developers can use these tools to monitor network requests, identify slow or inefficient requests, and optimize the code to improve network performance.

Overall, App Inspection is a valuable tool for Android developers. It provides a suite of tools for analyzing and debugging Android applications, identifying and fixing errors, and optimizing the code for improved performance and stability. By using App Inspection, developers can ensure that their Android applications are of the highest quality and provide the best possible user experience.

**6. Build Variants**

Build variants in Android Studio are different versions of an Android app that can be built from the same source code. They are typically used to create multiple versions of an app that target different device configurations or use cases. Build variants are configured in the build.gradle file and can be built and installed separately from each other. Here are some examples of how build variants can be used.

**Debug and Release Variants:** The most common use of build variants is to create a debug variant and a release variant of an app. The debug variant is used for testing and debugging the app during development, while the release variant is used for production and is optimized for performance and stability.

**Flavors:** Build variants can also be used to create different flavors of an app, which can have different features or configurations. For example, an app might have a free version and a paid version, or a version that targets tablets and a version that targets phones.

**Build Types:** Build variants can also be used to create different build types, which can have different build options or signing configurations. For example, an app might have a debug build type and a release build type, each with its own set of build options.

Overall, build variants are a powerful tool for Android developers. They allow developers to create different versions of an app from the same source code, which can save time and improve the quality of the app. By using build variants, developers can easily target different device configurations or use cases, create different versions of the app with different features or configurations, and optimize the app for performance and stability.

## 1.2 Purpose

Real-time communications (RTC) is any mode of telecommunications in which **all users can exchange information instantly or with negligible latency or transmission delays**. In this context, the term real-time is synonymous with live. In RTC, there is always a direct path between the source and the destination.

These applications **enable two-way communication that is more efficient than the one-way communication that is typical of nonreal-time applications**. This improves collaborations and overall communication within businesses. Response time. Real-time apps can respond faster to user input than traditional ones.

## 2.Problem Definition & Design Thinking

Problem Definition is the process of clearly defining and understanding a problem that needs to be solved. It involves identifying the root cause of the problem, its impact, and its scope. well understood before any solution is proposed. Design Thinking is a problem-solving approach that focuses on understanding the needs and perspectives of the user, and using that understanding to generate creative and effective solutions. It involves a series of iterative steps, including empathizing with the user, defining the problem, ideating potential solutions, prototyping, and testing. Design Thinking can be used to help with problem definition by encouraging a deep understanding of the problem and its impact on the user. By empathizing with the user and understanding their needs and perspective, designers can gain a better understanding of the problem and its scope. This understanding can then be used to generate more effective solutions that are more likely to meet the needs of the user.Overall, problem definition and Design Thinking are both important processes in the development of effective solutions. Problem definition ensures that the problem is well understood, while Design Thinking helps to generate creative and effective solutions that meet the needs of the user.

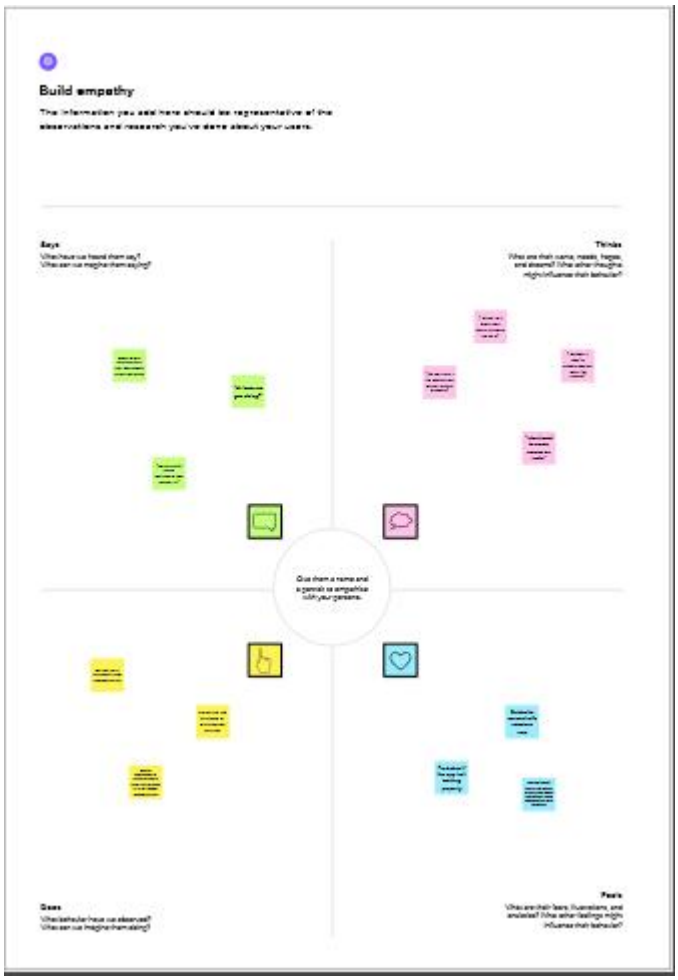
### 2.1 Empathy Map:

An empathy map is a tool that helps you gain a deeper understanding of your users by putting yourself in their shoes. It is a visual representation of what a user thinks, feels, sees, hears, and does in relation to a specific problem or situation. Let's say you are designing a chat app. To create an empathy map for this app, you would start by identifying your target users. For example, your target users could be young adults between the ages of 18-24 who use chat apps frequently to communicate with friends and family.

- Thinks: What are the user's thoughts and concerns when using a chat app? For example, they may worry about privacy and security or may be concerned about the app being too complicated to use.
- Feels: What are the user's emotions when using a chat app? For example, they may feel frustrated when the app crashes or may feel happy when they receive a message from a friend.
- Sees: What does the user see when using a chat app? For example, they may see a list of contacts or a chat window with a friend.

- ❖ Hears: What does the user hear when using a chat app? For example, they may hear a notification sound when they receive a message or may hear the sound of typing when their friend is responding.
- ❖ Does: What does the user do when using a chat app? For example, they may send messages, create group chats, or share photos and videos.

## Screenshot of Empathy Map:



**Ideation & Brainstroming Map:**

Ideation is the process of generating new ideas and solutions to a problem. It is an essential part of the design thinking process and involves a range of techniques and methods to help teams generate a large number of ideas quickly and effectively.

Brainstorming is a commonly used ideation technique that involves a group of people coming together to generate as many ideas as possible in a short amount of time. The goal of brainstorming is to generate a wide variety of ideas without judgment or evaluation. The focus is on quantity over quality.
To conduct a successful brainstorming session, there are a few key principles to follow:Encourage participation: Everyone in the group should be encouraged to contribute their ideas, no matter how small or seemingly insignificant.Defer judgment: Criticism or evaluation of ideas should be deferred until after the brainstorming session is complete.

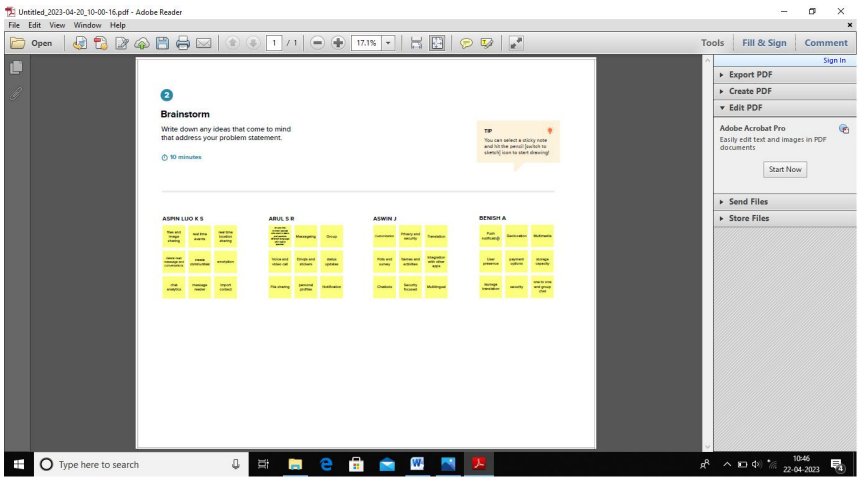
This creates a safe and supportive environment where participants feel comfortable sharing their ideas.Build on each other's ideas: Participants should be encouraged to build on the ideas of others, rather than simply coming up with new ideas.Stay focused on the topic:

The group should stay focused on the problem or challenge at hand, and all ideas should be related to the topic.Use visual aids: Using visual aids such as whiteboards, post-it notes, or mind maps can help to organize and stimulate the ideation process.

Once the brainstorming session is complete, the ideas generated can be evaluated and refined using other ideation techniques such as prioritization, clustering, or concept mapping. This helps to identify the most promising ideas and develop them further into viable solutions.

Overall, ideation and brainstorming are important processes in the design thinking approach, as they help to generate a wide range of ideas and solutions to a problem. By following the principles of brainstorming and using a variety of ideation techniques, teams can develop innovative and effective solutions to complex problems.

# Screenshot of Ideation & Brainstorming Map:




# 3. RESULT

### 3.1 Data Model:

A chat app data model can vary depending on the specific features and functionalities of the application, but here is a general overview of the key components:

**User:** A user represents a person who is using the chat app. It typically includes attributes such as name, email, password, profile picture, and other relevant information

**Conversation:** A conversation is a thread of messages between two or more users. It can be a one-on-one conversation or a group conversation. Each conversation has a unique identifier and can have multiple participants.

**Message:** A message is a piece of content that a user sends in a conversation. It can be text, images, videos, audio, or other types of media. Each message has a timestamp and can include metadata such as the sender and the recipient.

**Chat room:** A chat room is a virtual space where multiple users can join and chat with each other. It typically has a name or topic and can be public or private.

**Notifications:** Notifications are alerts that notify users of new messages, invitations to join a conversation or chat room, or other relevant events. They can be delivered via push notifications, email, or in-app notifications.

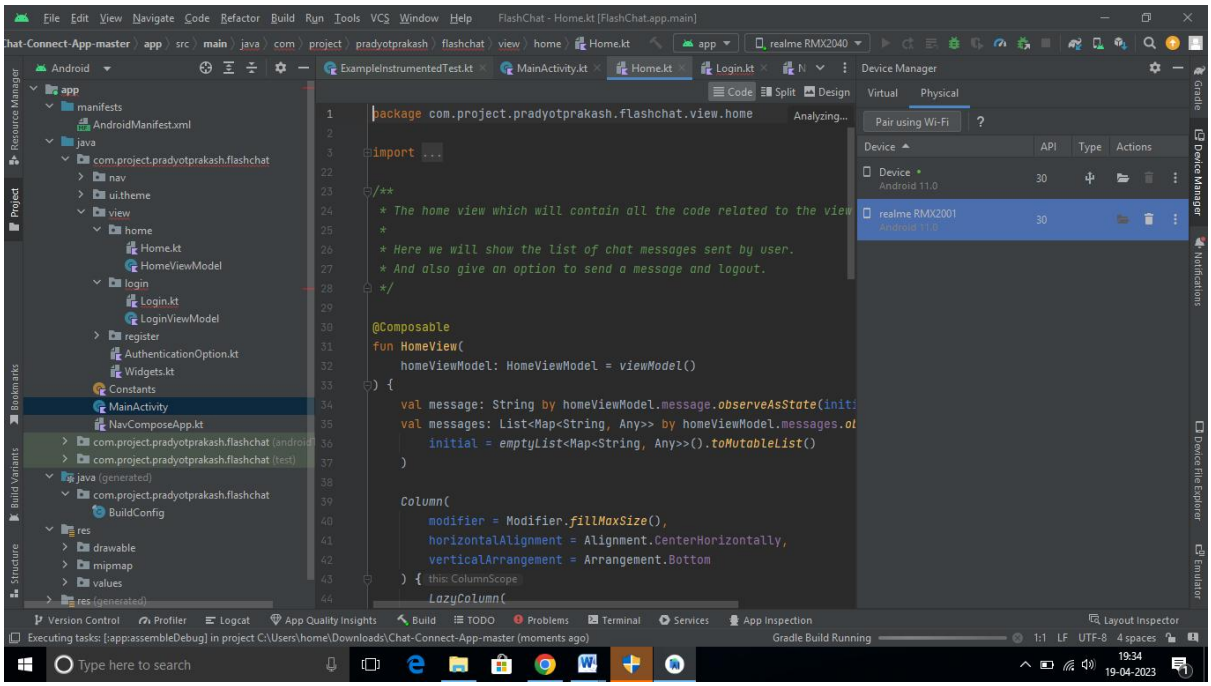
**Contacts:** Contacts are a list of users that a user has added as friends or frequently chats with. It can include attributes such as name, profile picture, and status.

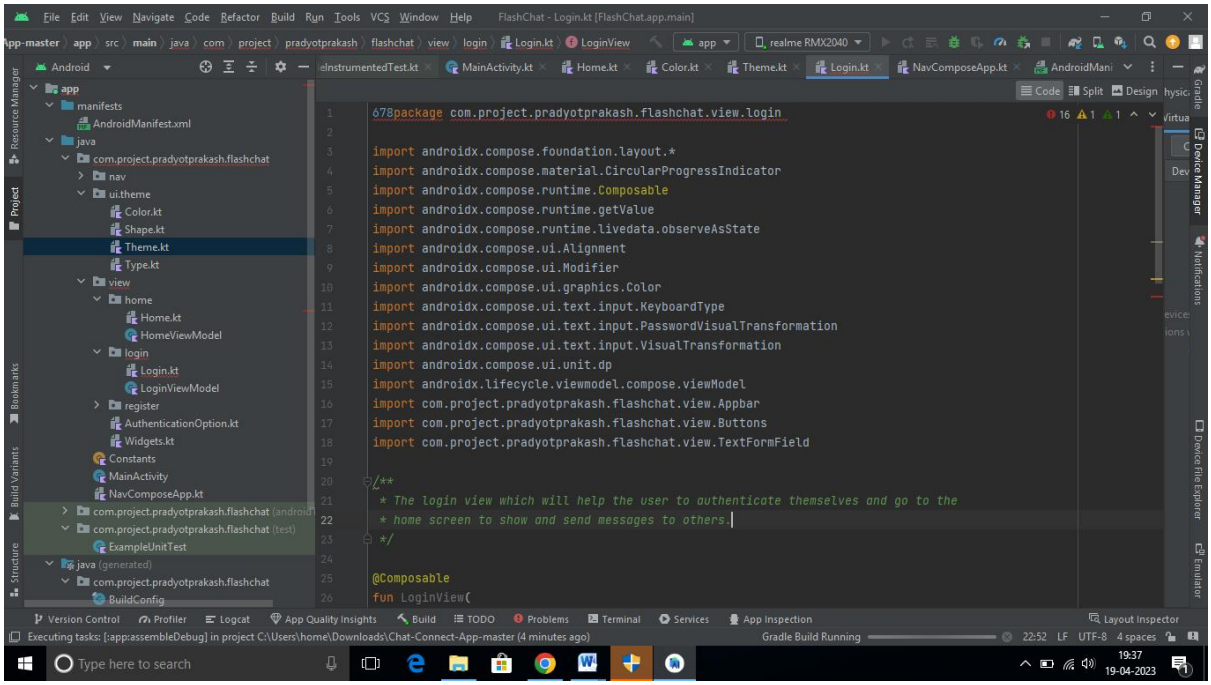
**Status:** A status is a message that a user can set to indicate their availability or mood. It can be text or an emoji and is visible to other users in the app.

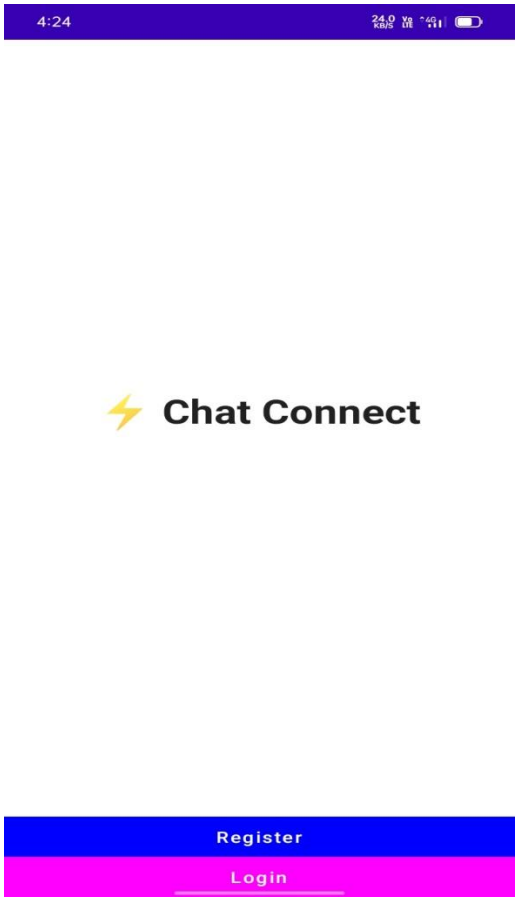
**Settings:** Settings are options that allow users to customize their app experience. It can include features such as notification preferences, privacy settings, and account management.

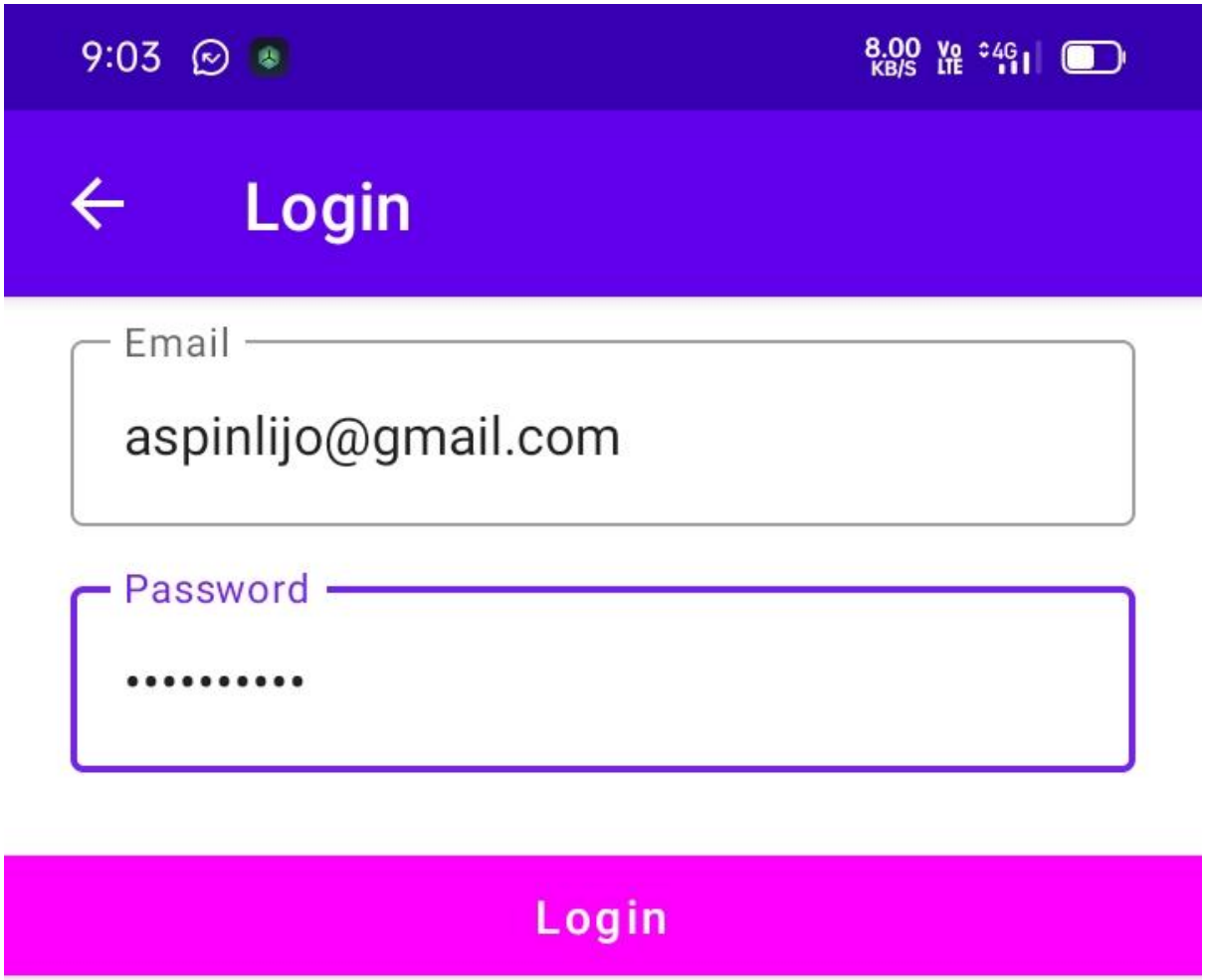
This is just a high-level overview, and there may be additional features or components depending on the specific chat app

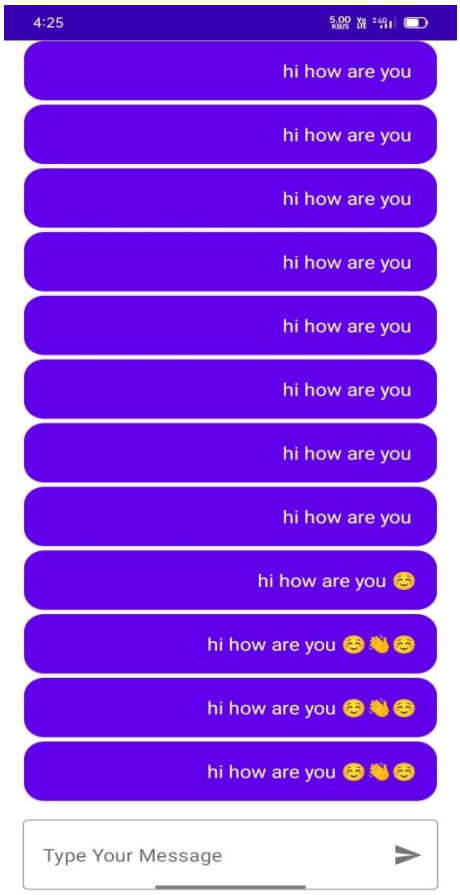
### 3.2 Activity & Screenshot


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
FlashChat - Home.kt [FlashChat.app.main]
ExampleInstrumentedTest.kt
MainActivity.kt
Home.kt
Login.kt
Code Split Design
Analyzing...
package com.project.pradyotprakash.flashchat.view.home
import ...
/**
* The home view which will contain all the code related to the view
*
* 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(initi
val messages: List<Map<String, Any>> by homeViewModel.messages.ol
initial = emptyList<Map<String, Any>>().toMutableList()
)
Column(
modifier = Modifier.fillMaxSize(),
horizontalAlignment = Alignment.CenterHorizontally,
verticalArrangement = Arrangement.Bottom
) { this: ColumnScope
LazyColumn(
Device Manager
Virtual
Physical
Pair using Wi-Fi
Device
API
Type
Actions
Device
Android 11.0
30
realme RMX2001
Android 11.0
30
Version Control
Profiler
Logcat
App Quality Insights
Build
TODO
Problems
Terminal
Services
App Inspection
Layout Inspector
Executing tasks: [:app:assembleDebug] in project C:\Users\home\Downloads\Chat-Connect-App-master (moments ago)
Gradle Build Running
1:1
LF
UTF-8
4 spaces
Type here to search
19:34
19-04-2023


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
FlashChat - Login.kt [FlashChat.app.main]
678package 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(
Version Control
Profiler
Logcat
App Quality Insights
Build
TODO
Problems
Terminal
Services
App Inspection
Layout Inspector
Executing tasks: [:app:assembleDebug] in project C:\Users\home\Downloads\Chat-Connect-App-master (4 minutes ago)
Gradle Build Running
Type here to search
19:37
19-04-2023


4:24
⚡ Chat Connect
Register
Login


9:03
8.00 KB/S
Login
Email
aspinlijo@gmail.com
Password
••••••••••
Login


4:25
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you 😊
hi how are you 😊👋😊
hi how are you 😊👋😊
hi how are you 😊👋😊
Type Your Message

## 4.ADVANTAGES & DISADVANTAGES

### 4.1 Advantages

Instant communication: Chatting apps allow you to communicate with others instantly, regardless of the distance between you.

Cost-effective: Most chatting apps are free to use, which makes them a cost-effective way to communicate with others, especially for long-distance communication.

Convenience: Chatting apps are very convenient to use, as they can be accessed from anywhere, at any time, as long as you have an internet connection.

Group communication: Chatting apps allow you to communicate with multiple people at the same time, making them ideal for group communication.

Rich media sharing: Chatting apps often support sharing of various types of media, including photos, videos, voice notes, and documents, making it easy to share information and collaborate with others.

### 4.2Disadvantages

Security and privacy concerns: Chatting apps can be vulnerable to security breaches and privacy violations, especially if they are not properly secured and encrypted.

Addiction: Chatting apps can be addictive, leading to overuse and a loss of productivity.

Misinformation: Chatting apps can be used to spread misinformation and fake news, which can be harmful to individuals and society as a whole.

Decreased social interaction: Chatting apps can lead to decreased face-to-face social interaction, which can have negative effects on mental health and well-being.

Distractions: Chatting apps can be distracting, leading to a loss of focus and reduced productivity, especially when used during work or study time.

## 5.APPLICATIONS

The areas where this solution can be applied

The solution of using chatting apps responsibly and balancing their benefits and drawbacks can be applied in many areas where chatting apps are commonly used. For example:

Business: Many businesses use chatting apps for communication and collaboration among employees, teams, and clients. By using these apps responsibly, businesses can enhance their productivity and efficiency.

Education: Chatting apps are commonly used in education to facilitate communication between students and teachers, as well as for collaborative projects. By using these apps responsibly, educators can enhance student learning and engagement.

Healthcare: Chatting apps can be used in healthcare for communication between doctors, nurses, and patients, as well as for telemedicine consultations. By using these apps responsibly, healthcare providers can improve patient outcomes and access to care.

Social Media: Chatting apps are widely used on social media platforms for personal communication and social networking. By using these apps responsibly, individuals can enhance their social connections while avoiding the negative impacts of addiction, misinformation, and decreased social interaction.

In all of these areas, the responsible use of chatting apps can lead to significant benefits while minimizing the risks associated with their use.

## 6.CONCLUSION

In conclusion, this work has examined the advantages and disadvantages of chatting apps and the importance of using them responsibly. Chatting apps have revolutionized communication by offering instant messaging, cost-effectiveness, convenience, group communication, and media sharing. However, they also present some risks, such as security and privacy concerns, addiction, misinformation, decreased social interaction, and distractions. To maximize the benefits of chatting apps while minimizing their potential negative impacts, it is crucial to use them responsibly and balance their benefits and drawbacks.

This solution of using chatting apps responsibly can be applied in various areas such as business, education, healthcare, and social media. By doing so, businesses can enhance their productivity and efficiency, educators can improve student learning and engagement, healthcare providers can improve patient outcomes and access to care, and individuals can enhance their social connections while avoiding the negative impacts of addiction, misinformation, and decreased social interaction.

Overall, the responsible use of chatting apps can lead to significant benefits, and it is up to individuals to use these apps in a way that serves their needs while being mindful of the potential risks.

## 7.FUTURE SCOPE

There are several enhancements that can be made in the future for chatting apps. Here are some ideas:

Voice and Video Calling: Voice and video calling can be added to the chat app, making it possible for users to make audio and video calls to their contacts within the app. This would add more functionality to the app and make it a more versatile communication tool.

End-to-End Encryption: End-to-end encryption can be added to the chat app to ensure that all conversations are private and secure. This will protect users' personal information and make the app more trustworthy.

AI Chatbots: Chatbots powered by artificial intelligence can be integrated into the chat app to assist users with various tasks, such as booking appointments or making reservations. This will provide a more personalized experience for users and save them time.

Group Chat Features: Group chat features can be enhanced to allow users to create sub-groups within larger groups. This would enable users to have more focused discussions with specific individuals or on specific topics.

Integration with Other Apps: Chat apps can be integrated with other apps, such as calendars or task management tools, making it easier for users to manage their schedules and tasks within the app.

Emojis and Stickers: More emojis and stickers can be added to the app to enhance the user experience and make conversations more fun and engaging.

Customization Options: Customization options can be added to the app, allowing users to customize the app's interface to their liking. This would give users a sense of ownership and make the app more personalized.

Better Search Functionality: The search functionality can be improved to make it easier for users to find past conversations, contacts, or messages.

Integration with Wearable Devices: Chat apps can be integrated with wearable devices, such as smartwatches, making it possible for users to send and receive messages without having to take out their phones.

Multi-Language Support: Multi-language support can be added to the app, making it more accessible to users who speak different languages. This would expand the app's user base and increase its global reach.

## 8.APPENDIX

### A. Source code

### Manitests:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.flashchat">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/quick"
        android:label="Chat Connect"
        android:roundIcon="@drawable/quick"
        android:supportsRtl="true"
        android:theme="@style/Theme.FlashChat">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Chat Connect"
            android:theme="@style/Theme.Chatconnect.NoActionBar">
            <intent-filter>
```

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

**Main Activity:**

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

## NavCompose:

```
package com.Chatconnect

import androidx.compose.runtime.Composable

import androidx.compose.runtime.remember

import androidx.navigation.compose.NavHost

import androidx.navigation.compose.composable

import androidx.navigation.compose.rememberNavController

import com.google.firebase.auth.FirebaseAuth

import com.flashchat.nav.Action

import com.flashchat.nav.Destination.AuthenticationOption

import com.flashchat.nav.Destination.Home

import com.flashchat.nav.Destination.Login

import com.flashchat.nav.Destination.Register

import com.flashchat.ui.theme. Chatconnect Theme
```

```
import com.flashchat.view.AuthenticationView
import com.flashchat.view.home.HomeView
import com.flashchat.view.login.LoginView
import com.flashchat.view.register.RegisterView
@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:

```
package com.Chatconnect

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

    const val MESSAGES = "messages"
```

```
    const val MESSAGE = "message"
    const val SENT_BY = "sent_by"
    const val SENT_ON = "sent_on"
    const val IS_CURRENT_USER = "is_current_user"
}
```

## Navigation:

```
package com. Chatconnect.nav

import androidx.navigation.NavHostController

import com.Chatconnect.nav.Destination.Home
```

```
import com.Chatconnect.nav.Destination.Login
import com.Chatconnect.nav.Destination.Register
object Destination {
    const val AuthenticationOption = "authenticationOption"
    const val Register = "register"
    const val Login = "login"
    const val Home = "home"
}
class Action(navController: NavHostController) {
    val home: () -> Unit = {
        navController.navigate(Home) {
            popUpTo(Login) {
                inclusive = true
            }
            popUpTo(Register) {
                inclusive = true
            }
        }
    }
    val login: () -> Unit = { navController.navigate(Login) }
    val register: () -> Unit = { navController.navigate(Register) }
    val navigateBack: () -> Unit = { navController.popBackStack() }
}
```

**view package:**

## AuthenticationOption:

```
package com. Chatconnect.view
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import com.flashchat.ui.theme. Chatconnect Theme
@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {
chatconnect Theme {
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.Red)
            Buttons(title = "Login", onClick = login, backgroundColor = Color.Green)
        }
    }
  }
}
```

## Widgets:

```
package com. Chatconnect.view

import androidx.compose.foundation.layout.fillMaxHeight

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.foundation.text.KeyboardOptions

import androidx.compose.material.*

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.ArrowBack

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.text.input.KeyboardType

import androidx.compose.ui.text.input.VisualTransformation

import androidx.compose.ui.text.style.TextAlign

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp
```

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

@Composable
fun Buttons(title: String, onClick: () -> Unit, backgroundColor: Color) {
    Button(
        onClick = onClick,
        colors = ButtonDefaults.buttonColors(
            backgroundColor = backgroundColor,
            contentColor = Color.White
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = RoundedCornerShape(0),
    ) {
        Text(
            text = title
        )
    }
```

```
}
@Composable
fun Appbar(title: String, action: () -> Unit) {
    TopAppBar(
        title = {
            Text(text = title)
        },
        navigationIcon = {
            IconButton(
                onClick = action
            ) {
                Icon(
                    imageVector = Icons.Filled.ArrowBack,
                    contentDescription = "Back button"
                )
            }
        }
    )
}
@Composable
fun TextFormField(value: String, onValueChange: (String) -> Unit, label: String, keyboardType: KeyboardType, visualTransformation: VisualTransformation) {
    OutlinedTextField(
        value = value,
```

```
        onValueChange = onValueChange,
        label = {
            Text(
                label
            )
        },
        maxLines = 1,
        modifier = Modifier
            .padding(horizontal = 20.dp, vertical = 5.dp)
            .fillMaxWidth(),
        keyboardOptions = KeyboardOptions(
            keyboardType = keyboardType
        ),
        singleLine = true,
        visualTransformation = visualTransformation
    )
}
@Composable
fun SingleMessage(message: String, isCurrentUser: Boolean) {
    Card(
        shape = RoundedCornerShape(16.dp),
        backgroundColor = if (isCurrentUser) MaterialTheme.colors.primary else
Color.White
    ) {
        Text(
```

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

## Home:

```
package com. Chatconnect.view.home
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardOptions
```

```
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Send
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import com. Chatconnect.Constants
import com. Chatconnect.view.SingleMessage
@Composable
fun HomeView(
    homeViewModel: HomeViewModel = viewModel()
) {
    val message: String by  homeViewModel.message.observeAsState(initial = "")
    val messages: List<Map<String, Any>> by
homeViewModel.messages.observeAsState(
        initial = emptyList<Map<String, Any>>().toMutableList()
    )
    Column(
        modifier = Modifier.fillMaxSize(),
```

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


                SingleMessage(
                    message = message[Constants.MESSAGE].toString(),
                    isCurrentUser = isCurrentUser
                )
            }
        }
        OutlinedTextField(
            value = message,
            onValueChange = {
                homeViewModel.updateMessage(it)
```

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

```
                )
            }
        }
    )
  }
}
```

## HomeViewModel class:

```
package com. chatconnect.view.home

import android.util.Log

import androidx.lifecycle.LiveData

import androidx.lifecycle.MutableLiveData

import androidx.lifecycle.ViewModel

import com.google.firebase.auth.ktx.auth

import com.google.firebase.firestore.ktx.firestore

import com.google.firebase.ktx.Firebase

import com. Chatconnect.Constants

import java.lang.IllegalArgumentException

class HomeViewModel : ViewModel() {

    init {

        getMessages()

    }

    private val _message = MutableLiveData("")

    val message: LiveData<String> = _message

    private var _messages = MutableLiveData(emptyList<Map<String,
Any>>().toMutableList())

    val messages: LiveData<MutableList<Map<String, Any>>> = _messages

    fun updateMessage(message: String) {

        _message.value = message
```

```
    }
    fun addMessage() {
        val message: String = _message.value ?: throw
IllegalArgumentException("message empty")
        if (message.isNotEmpty()) {
            Firebase.firestore.collection(Constants.MESSAGES).document().set(
                hashMapOf(
                    Constants.MESSAGE to message,
                    Constants.SENT_BY to Firebase.auth.currentUser?.uid,
                    Constants.SENT_ON to System.currentTimeMillis()
                )
            ).addOnSuccessListener {
                _message.value = ""
            }
        }
    }
    private fun getMessages() {
        Firebase.firestore.collection(Constants.MESSAGES)
            .orderBy(Constants.SENT_ON)
            .addSnapshotListener { value, e ->
                if (e != null) {
                    Log.w(Constants.TAG, "Listen failed.", e)
                    return@addSnapshotListener
                }
                val list = emptyList<Map<String, Any>>().toMutableList(
```

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

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

## Login:

```
package com. Chatconnect.view.login

import androidx.compose.foundation.Image

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.draw.alpha

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale
```

```
import androidx.compose.ui.res.painterResource
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. Chatconnect.view.Appbar
import com. chatconnect.view.Buttons
import com. chatconnect.view.TextFormField
@Composable
fun LoginView(
    home: () -> Unit,
    back: () -> Unit,
    loginViewModel: LoginViewModel = viewModel()
) {
    val email: String by loginViewModel.email.observeAsState("")
    val password: String by loginViewModel.password.observeAsState("")
    val loading: Boolean by loginViewModel.loading.observeAsState(initial =
false)
    val imageModifier = Modifier
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
```

```
Box(
    contentAlignment = Alignment.Center,
    modifier = Modifier.fillMaxSize()
) {
    if (loading) {
        CircularProgressIndicator()
    }
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Top
    ) {
        Appbar(
            title = "Login",
            action = back
        )
        TextFormField(
            value = email,
            onValueChange = { loginViewModel.updateEmail(it) },
            label = "Email",
            keyboardType = KeyboardType.Email,
            visualTransformation = VisualTransformation.None
        )
        TextFormField(
```

```
                value = password,
                onValueChange = { loginViewModel.updatePassword(it) },
                label = "Password",
                keyboardType = KeyboardType.Password,
                visualTransformation = PasswordVisualTransformation()
            )
            Spacer(modifier = Modifier.height(20.dp))
            Buttons(
                title = "Login",
                onClick = { loginViewModel.loginUser(home = home) },
                backgroundColor = Color.Magenta
            )
        }
    }
}
```

## LoginViewModel:

```
package com. Chatconnect.view.login
```

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException
class LoginViewModel : ViewModel() {
    private val auth: FirebaseAuth = Firebase.auth
    private val _email = MutableLiveData("")
    val email: LiveData<String> = _email
    private val _password = MutableLiveData("")
    val password: LiveData<String> = _password
    private val _loading = MutableLiveData(false)
    val loading: LiveData<Boolean> = _loading
    fun updateEmail(newEmail: String) {
        _email.value = newEmail
    }
    fun updatePassword(newPassword: String) {
        _password.value = newPassword
    }
    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:

```
package com. Chatconnect.view.register

import androidx.compose.foundation.Image

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.draw.alpha

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

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. Chatconnect.view.Appbar

import com. Chatconnect.view.Buttons
```

```
import com. Chatconnect.view.TextFormField

@Composable

fun RegisterView(

    home: () -> Unit,

    back: () -> Unit,

    registerViewModel: RegisterViewModel = viewModel()

) {

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

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

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

    val imageModifier = Modifier


    Box(

        contentAlignment = Alignment.Center,

        modifier = Modifier.fillMaxSize()

    ) {

        if (loading) {

            CircularProgressIndicator()

        }

        Column(

            modifier = Modifier.fillMaxSize(),

            horizontalAlignment = Alignment.CenterHorizontally,

            verticalArrangement = Arrangement.Top

        ) {
```

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

```
        }
    }
}
```

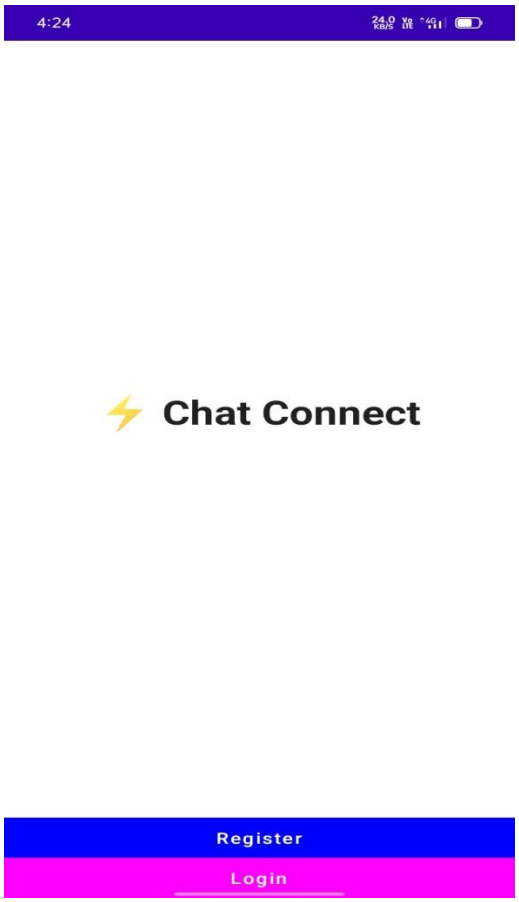
## RegisterViewModel class:

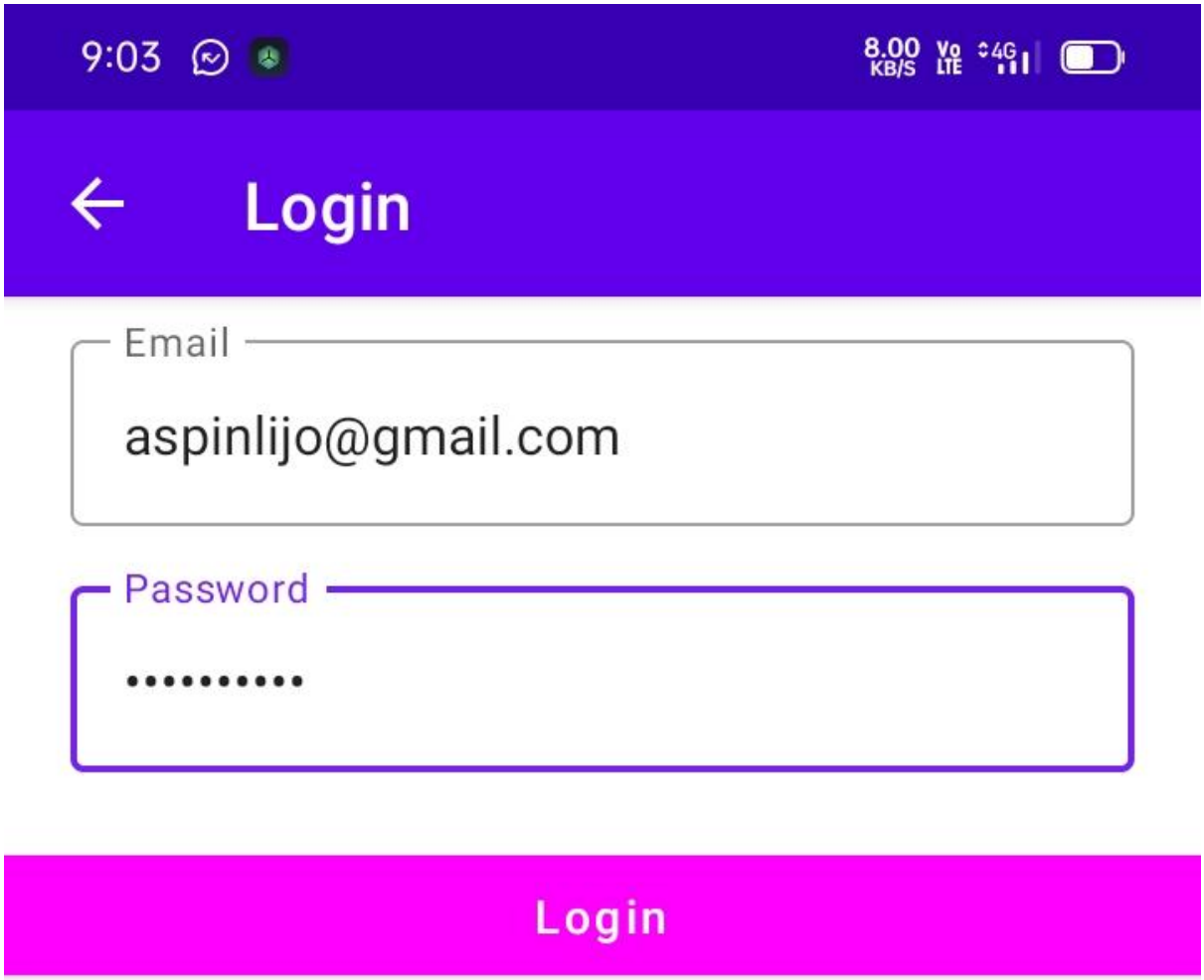
```
package com. Chatconnect.view.register

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import java.lang.IllegalArgumentException
class RegisterViewModel : ViewModel() {
```

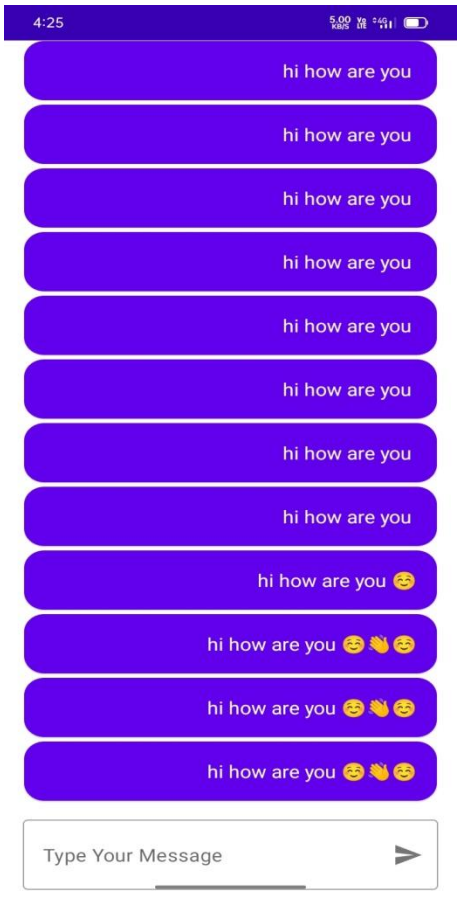
```
private val auth: FirebaseAuth = Firebase.auth

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

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

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

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

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

// Register user
fun registerUser(home: () -> Unit) {
    if (_loading.value == false) {
```

```
        val email: String = _email.value ?: throw
IllegalArgumentException("email expected")
        val password: String =
            _password.value ?: throw IllegalArgumentException("password
expected")

        _loading.value = true

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

## Output:




9:03
8.00 KB/S
Login
Email
aspinlijo@gmail.com
Password
••••••••••
Login


4:25
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you
hi how are you 😊
hi how are you 😊👋😊
hi how are you 😊👋😊
hi how are you 😊👋😊
Type Your Message