

**“CHATCONNECT - A REAL-TIME CHAT AND
COMMUNICATION APP”**

PROJECT REPORT

Submitted by

SHIJIN S (20203111506245)

SHIJO BLESSWIN S P (20203111506246)

RUBAN E (20203111506243)

ROLESON J G (20203111506242)

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.Phil



**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“**CHAT CONNECT - A REAL-TIME CHAT AND COMMUNICATIONAPP**” 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

Date:20-04-2023

Shijin S(20203111506245)

Shijo blessing S P (20203111506246)

Ruban E (20203111506243)

Roleson J G (20203111506242)

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.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 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.

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.

Furthermore, we want to extend special thanks towards our Naan Mudhalvan Smart internz team as well because without their resources then none of what is seen now could have been possible in terms of creative or intellectual development.

CONTENTS

PREFACE

1 INTRODUCTION

1.1 Overview

1.2 Purpose

2 PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

2.2 Ideation & Brainstorming Map

3 RESULT

3.1 Data Model

3.2 Activity & Screenshot

4 ADVANTAGES & DISADVANTAGE

4.1 Advantages

4.2 Disadvantage

5 APPLICATIONS

6 CONCLUSION

7 FUTURE SCOPE

8 APPENDIX

8.1. Source Code

1 INTRODUCTION:

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. 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 with 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 analyse 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 behaviour 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:

Secure Mobile Chat Requirements In this section, we propose a set of requirements to make secure chat application: Password stored on the chat server should be encrypted. Providing either secure session or TLS. Secure session is a unique key for each session. Ensures that communication is with the right person and no man. Local storage must be protected by encryption Messages are not stored on the chat server but stored on the user's device. It is not allowed to exchange messages if they are not friends. The proposed architecture is designed to be Client-Server chat application. In client side, when a user sets up the application, the user either selects registration or log-in. In server side, the chat server consists of users' server and a message server. User's server that manages user's credentials. Message server handles messages between users by using Firebase Cloud Messaging (FCM). If the recipient is offline, the messages will be stored temporarily on the FCM queue for a specific period of time, and when

recipient becomes online these messages are forwarded to him then deleted from the queue

The purpose of a chat app is to facilitate communication and interaction between users in real-time or near real-time through digital devices such as smartphones, tablets, or computers. Chat apps provide a platform for individuals or groups to exchange messages, share information, engage in discussions, and express thoughts, ideas, or emotions.

Some common purposes of chat apps include:

Socializing: Chat apps are often used for casual conversations and socializing with friends, family, or acquaintances. Users can chat, share multimedia content, and express themselves using emojis, stickers, or GIFs.

Collaboration: Chat apps are widely used for professional communication and collaboration among colleagues, teams, or business partners. Users can exchange work-related information, discuss projects, share files, and coordinate tasks in real-time.

Customer support: Many businesses use chat apps as a channel for customer support, allowing users to chat with customer service representatives to resolve inquiries, provide feedback, or seek assistance with products or services.

Networking: Chat apps can be used for networking purposes, allowing users to connect with like-minded individuals, join interest-based communities, or participate in group discussions related to their hobbies, interests, or professional fields.

Education: Chat apps are also used in educational settings for communication between students and teachers, facilitating discussions, sharing course materials, and providing a platform for remote learning.

Entertainment: Some chat apps are specifically designed for entertainment purposes, such as online gaming communities, where users can chat with fellow gamers, form teams, and engage in multiplayer games.

Overall, the purpose of a chat app is to enable communication, collaboration, and interaction among users for various personal, social, professional, educational, or entertainment-related purposes.

Chat apps have become an essential part of modern communication, allowing users to interact in real-time and exchange messages, media, and other content. Android Studio, a popular integrated development environment (IDE) for Android app development, offers a powerful platform for creating chat apps that run on Android devices. The purpose of this chat app development in Android Studio is to provide a seamless and user-friendly communication experience for users on the Android platform.

A chat app developed in Android Studio can have various features and functionality, depending on the requirements and goals of the app. Some common features of a chat app may include:

User Registration and Authentication: Users can create accounts, login, and authenticate their identities to use the chat app securely.

Contacts and Friends List: Users can manage their contacts and friends list, add or remove friends, and search for other users to connect with.

Real-time Messaging: Users can send and receive text messages in real-time, allowing for instantaneous communication.

Multimedia Sharing: Users can share multimedia content such as images, videos, and audio messages with each other.

Group Chat: Users can create and participate in group chats, allowing for communication with multiple users simultaneously.

Notifications: Users can receive notifications for new messages, friend requests, or other important events in the app.

Online/Offline Status: Users can see the online/offline status of their contacts, indicating whether they are currently available for chat.

Message History: Users can view and manage their message history, allowing them to access past conversations.

Emojis, Stickers, and GIFs: Users can express themselves using emojis, stickers, and GIFs to add emotions and personality to their messages.

Customization: Users can customize their profile, chat settings, and notifications according to their preferences

The purpose of a chat app developed in Android Studio can vary depending on the specific use case and target audience. However, some common purposes and benefits of a chat app in Android Studio include:

Communication and Social Interaction: The primary purpose of a chat app is to facilitate communication and social interaction among users in real-time, allowing them to connect, engage, and share information or emotions.

Convenience and Mobility: Chat apps developed in Android Studio provide users with a convenient and mobile way to communicate using their Android devices, allowing them to stay connected and chat on the go.

Collaborative Communication: Chat apps can be used for collaborative communication in a professional or business setting, enabling teams or groups to exchange messages, share files, and coordinate tasks in real-time.

User Engagement and Retention: A well-designed chat app with engaging features such as multimedia sharing, emojis, stickers, and GIFs can help increase user engagement and retention, leading to a more loyal and active user base.

Enhanced User Experience: A chat app developed in Android Studio can provide a seamless and user-friendly experience, with features such as notifications, message history, online/offline status, and customization options, enhancing the overall user experience.

Networking and Community Building: Chat apps can be used for networking purposes, allowing users to connect with like-minded individuals, join interest-based communities, or participate in group discussions related to their hobbies, interests, or professions.

Customer Support and Service: Chat apps can serve as a customer support channel, enabling users to seek assistance, resolve inquiries, and provide feedback on products or services, leading to improved customer satisfaction.

In today's fast-paced world, staying connected with people has become easier than ever before. Chat apps, also known as messaging apps, have revolutionized the way we communicate with each other. They offer a convenient and efficient way to stay in touch with friends, family, colleagues, and customers. In this article, we'll discuss the purpose of a chat app in detail and explore its benefits.

Facilitating Communication:

The primary purpose of a chat app is to facilitate communication between individuals or groups. It allows people to send and receive text messages in real-time, making it easier to stay connected with each other. Chat apps are especially useful for people who live far apart and cannot meet in person regularly. With a chat app, they can stay in touch with each other no matter where they are.

Personal and Professional Use:

Chat apps can be used for both personal and professional purposes. People use chat apps to communicate with friends and family, plan events, share photos and videos, and more. On the other hand, businesses use chat apps to communicate

with their employees, collaborate on projects, provide customer support, and conduct transactions.

Convenience:

Chat apps are incredibly convenient to use. They are available on multiple devices, including smartphones, tablets, and computers, and can be accessed from anywhere with an internet connection. This convenience makes it easy for people to stay in touch with each other, regardless of their location or time zone.

Features:

Chat apps offer a wide range of features that enhance the user experience. For example, voice and video calls allow people to have real-time conversations, even if they cannot meet in person. Group chats allow multiple people to communicate with each other at the same time. Emojis and stickers add fun and personality to messages. File sharing allows people to exchange documents, photos, and videos easily.

Security:

Chat apps are designed to be secure, protecting the privacy of users. Messages are encrypted, making it difficult for anyone to intercept them. Additionally, chat apps often have features such as two-factor authentication, passcodes, and fingerprint scanning, which add an extra layer of security.

Business Purposes:

Chat apps are particularly useful for businesses. They can be used for internal communication, collaboration, and project management. They also provide a

quick and easy way to provide customer support, allowing businesses to respond to inquiries in real-time.

Global Reach:

Chat apps are used all over the world, making them a powerful tool for reaching a global audience. This is particularly useful for businesses that want to expand their customer base and reach new markets.

Customization:

Chat apps can be customized to suit individual needs. For example, businesses can create custom chat apps with features that meet their specific needs. This customization allows chat apps to be tailored to individual users, making them more useful and efficient.

Integration:

Chat apps can be integrated with other tools and services, such as email, calendars, and social media platforms. This integration allows people to stay organized and manage their communication channels more effectively.

Cost-effective:

Chat apps are often free to use, making them a cost-effective communication tool. Additionally, businesses can save money on travel and phone expenses by using chat apps for internal communication and collaboration.

2PROBLEM 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.

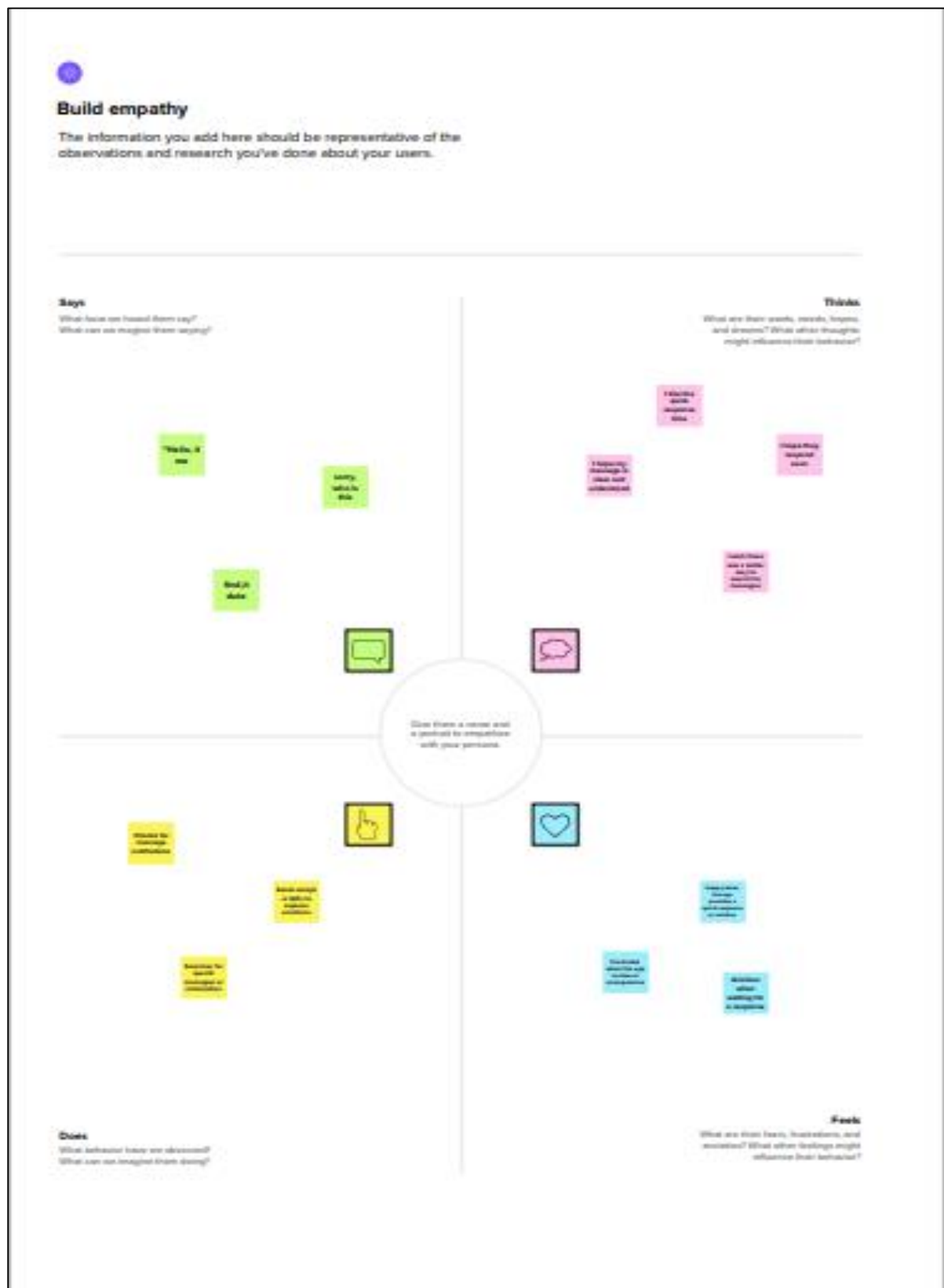
2.1 Empathy Map:

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:



2.2 Ideation & Brainstorming 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:

2

Brainstorm

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

10 minutes

TIP

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

SHIJIN S	SHIJO BLESSWIN S P	ROLESON J G	RUBAN E
Easy development and hosting	AR-based effects	sign up to join chat groups.	Synchronized cross platform app
It enables employees from across the world to communicate with each other 24/7 and share ideas	User presence indication	Storage of past messages	the chances of misunderstanding or misinterpretation are high.
Telegram is one of the first messengers to launch its own bot platform	Ability to interpret	unread message counts	Message scheduling
Leading data security	Ability to package and send a message	List of contacts with easy contact import and editing functionality	Telegram is one of the first messengers to launch its own bot platform
chat is a way of communication and interaction in real time on the Internet	List of contacts with easy contact import and editing functionality	Dis-to-Bot Encryption is a system of communication where particularly communicating users can read the messages	It allows employees to enjoy flawless
It enables employees from across the world to communicate with each other 24/7 and share ideas	uninterrupted connectivity using any device of their choice.	AI-powered chatbots	Collaboration tools
Message text editing field with keyboard	Leading mobile application management	Professional administration	Synchronized cross platform app
User presence indication	uninterrupted connectivity using any device of their choice	There are many types of chats – for example, ongoing or scheduled for a certain time	User presence indication
A block chain based messaging system	Chat, also known as online chat or internet chat	Leading data protection	A Tor based messaging system

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.

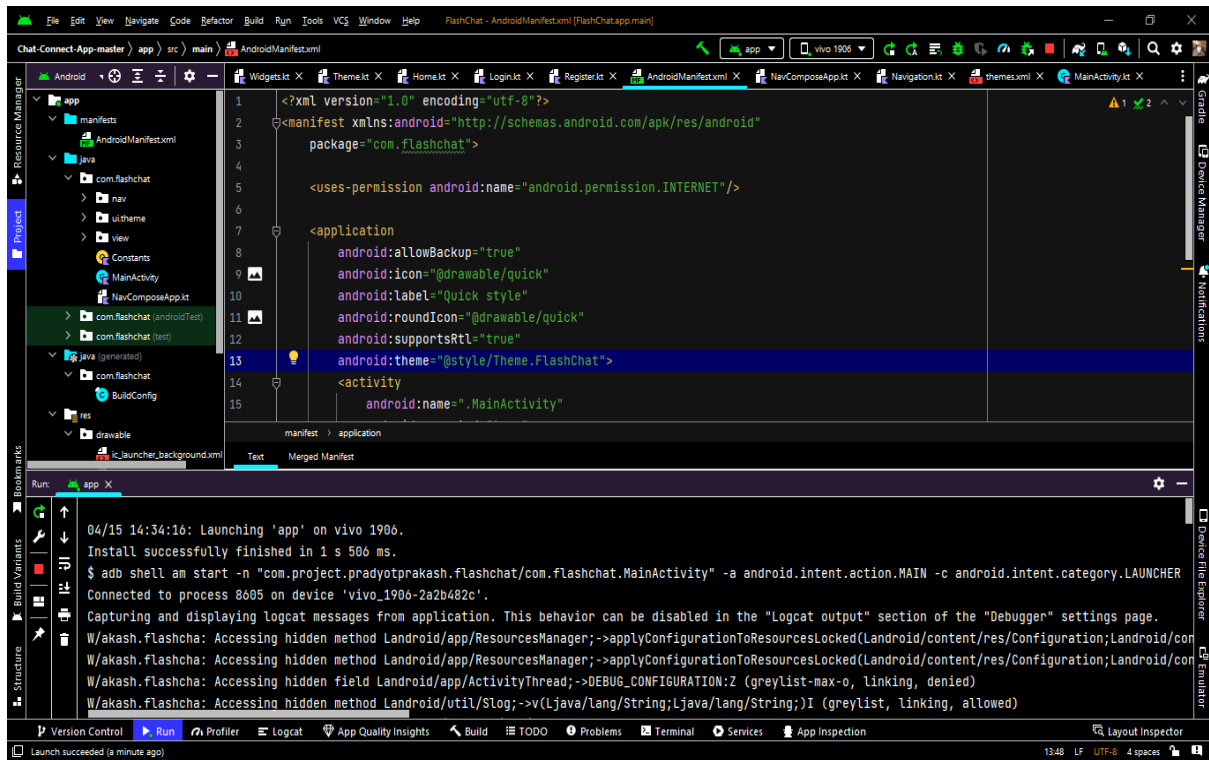
Contacts: Contacts are a list of users that a user has added as friends or frequently chats with. It can include attributes such as name, profile picture, and status.

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.

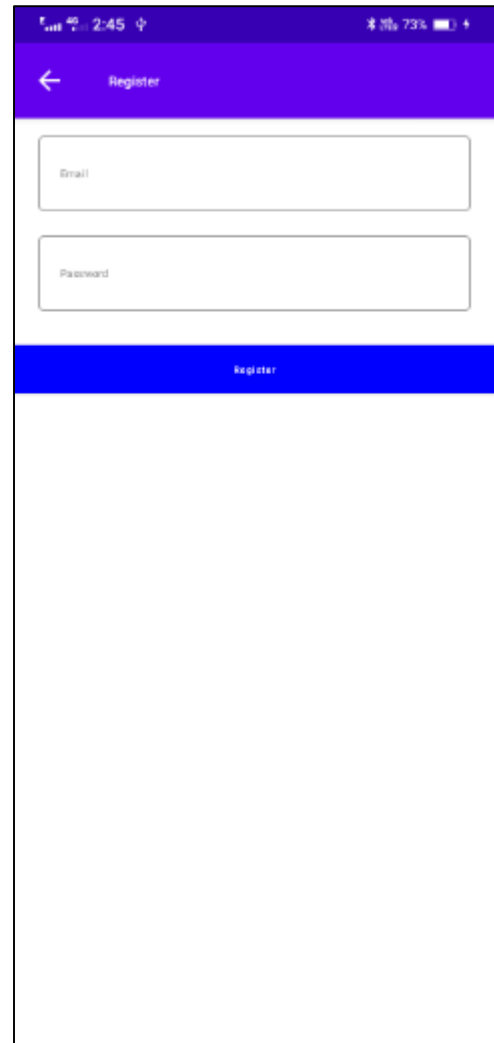
Settings: Settings are options that allow users to customize their app experience. It can include features such as notification preferences, privacy settings, and account management.

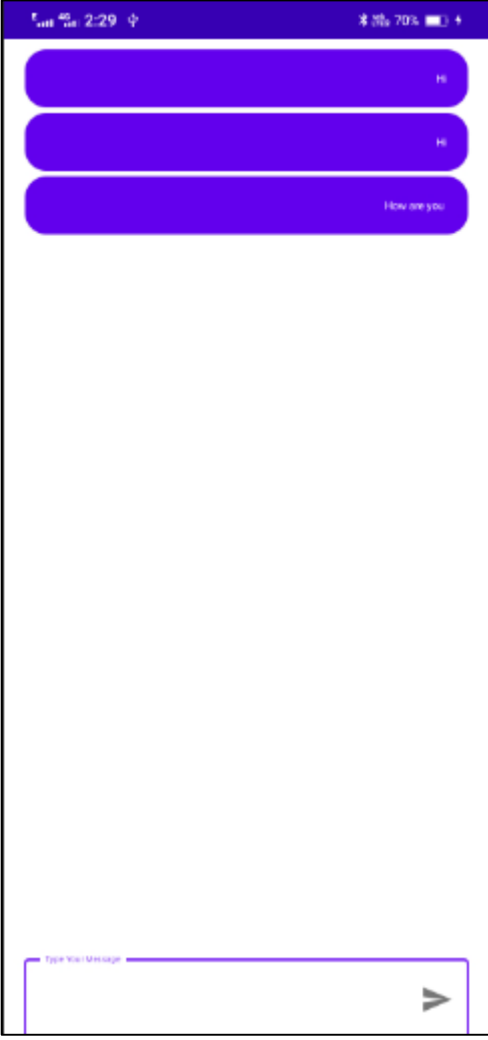
This is just a high-level overview, and there may be additional features or components depending on the specific chat app.

3.2 Activity & Screenshot









4 ADVANTAGES & DISADVANTAGE:

4.1 Advantages:

Real-time chat apps developed using Android Studio have become increasingly popular in recent years due to their ability to facilitate instant communication and connectivity. In this section, we will explore the advantages of real-time chat apps and how they can benefit users in various ways.

Instant Communication: One of the primary advantages of a real-time chat app is the ability to communicate instantly with other users. Users can send text messages, images, videos, voice messages, and even make voice or video calls in real-time, allowing for quick and efficient communication. This can be particularly beneficial in situations where immediate communication is required, such as in a professional setting, during emergencies, or when staying connected with loved ones. Real-time chat apps enable seamless and prompt communication, bridging the gap of time and distance, and enhancing connectivity among users.

Easy and Convenient Connectivity: Real-time chat apps provide easy and convenient connectivity among users. Users can connect with friends, family, colleagues, or even strangers from around the world with just a few taps on their smartphones. This convenience allows users to stay connected with their social and professional networks effortlessly, regardless of their location. Real-time chat apps also support group chats, enabling users to communicate with multiple contacts simultaneously, which can be particularly useful for team collaborations or group discussions. This easy and convenient connectivity fosters communication and collaboration, improving productivity, and enhancing relationships.

Rich Media Sharing: Real-time chat apps allow users to share a wide range of media, including images, videos, and voice messages. This rich media sharing capability enables users to express themselves more creatively and vividly, enhancing the overall communication experience. Users can share important information, memorable moments, or express their emotions through media sharing, making communication more engaging and expressive. Real-time chat apps also support file sharing, allowing users to send and receive documents, PDFs, or other files, making it a convenient tool for professional communication and collaboration.

Privacy and Security: Privacy and security are crucial considerations in any communication app, and real-time chat apps are no exception. Most real-time chat apps developed using Android Studio offer end-to-end encryption, ensuring that messages and media shared between users are secure and private. This encryption prevents unauthorized access or interception of messages, protecting user privacy and ensuring the confidentiality of conversations. Additionally, real-time chat apps often provide features such as message deletion, message forwarding, and blocking or reporting users, giving users control over their conversations and ensuring a safe and secure communication environment.

Real-time Interaction and Feedback: Real-time chat apps provide instant feedback and interaction, which can be beneficial in various scenarios. For example, in a customer support or service setting, real-time chat apps allow users to communicate with customer service representatives in real-time, enabling prompt responses and issue resolutions. Real-time chat apps can also be used in educational settings, facilitating real-time interactions between teachers and students or among classmates, enhancing the learning experience. Real-time feedback and interaction enable users to exchange information and ideas quickly, making communication more effective and efficient.

Customization and Personalization: Real-time chat apps developed using Android Studio often provide customization and personalization features, allowing users to tailor the app according to their preferences. Users can customize their profiles, set up personalized chat settings, choose notification preferences, and even personalize the overall appearance of the app. This customization and personalization ability enable users to create a personalized communication experience, making the app more enjoyable and user-friendly. Real-time chat apps that offer customization and personalization features can attract more users and enhance user satisfaction.

4.2 Disadvantage:

While real-time chat apps developed using Android Studio offer numerous advantages, they also come with certain disadvantages or challenges that users and developers may encounter. In this section, we will explore the potential drawbacks of real-time chat apps and the limitations they may have.

Reliance on Internet Connectivity: One of the primary disadvantages of real-time chat apps is their reliance on internet connectivity. Real-time chat apps require a stable internet connection to function properly, and users may face issues in areas with poor or no internet connectivity. This can result in messages not being delivered or received in real-time, and users may experience delays or disruptions in communication. Additionally, in situations where there is limited or no internet connectivity, real-time chat apps may not function at all, limiting their usability.

Privacy and Security Risks: Real-time chat apps can pose privacy and security risks, both for users and developers. Despite offering end-to-end encryption in many cases, there is still a potential risk of messages or media being intercepted, hacked, or leaked. Users may also face privacy concerns related to the collection of personal data, tracking of conversations, or profiling for targeted advertising. Developers may face challenges in ensuring the security of user data, protecting against data breaches, or complying with data privacy regulations. The potential privacy and security risks associated with real-time chat apps require careful consideration and implementation of robust security measures.

Information Overload: Real-time chat apps can result in information overload for users, especially in group chats or busy communication environments. Users may receive a high volume of messages, notifications, or media, leading to difficulty in managing and organizing the conversations. This can result in important messages being missed or overlooked, and users may feel

overwhelmed or stressed with the constant influx of information. Managing the flow of information and organizing conversations can be a challenge in real-time chat apps, and users may need to develop effective strategies to cope with information overload.

Miscommunication and Misinterpretation: Real-time chat apps rely solely on written or digital communication, which can lead to miscommunication or misinterpretation of messages. Unlike face-to-face communication, real-time chat apps lack non-verbal cues, tone of voice, or body language, which can often convey additional meaning or context. Users may misinterpret messages, leading to misunderstandings, conflicts, or misaligned expectations. Additionally, real-time chat apps often involve fast-paced conversations with quick responses, which can result in hasty or poorly composed messages, further increasing the risk of miscommunication. Users need to be cautious about the limitations of written communication in real-time chat apps and ensure clarity in their messages.

Distractions and Addiction: Real-time chat apps can be addictive and lead to distractions, both in personal and professional settings. Users may find themselves constantly checking their chat app for new messages or notifications, leading to distractions from other important tasks or activities. This can result in decreased productivity, reduced focus, and even impact mental health and well-being. Real-time chat apps can also lead to social media addiction, where users spend excessive time chatting with friends or strangers, neglecting real-life interactions or relationships. Managing the balance between real-time chat app usage and other important aspects of life can be challenging, and users need to be mindful of the addictive nature of these apps.

Quality of Communication: Real-time chat apps may not always provide the same quality of communication as face-to-face or voice communication. Text-based communication can sometimes lack the nuances, emotions, or expressions

of spoken communication, leading to a less rich and meaningful communication experience.

5APPLICATIONS:

Real-time chat apps developed using Android Studio have found widespread applications in various domains, ranging from personal communication to professional collaboration. These apps have revolutionized the way people interact and communicate, providing instant and convenient communication channels. In this section, we will explore the diverse applications of real-time chat apps and how they have been used to enhance communication and collaboration in different settings.

Personal Communication: Real-time chat apps have become an integral part of personal communication, allowing users to stay connected with friends, family, and acquaintances in real-time. These apps provide a platform for instant messaging, voice calls, and video calls, enabling users to communicate with loved ones irrespective of their location. Real-time chat apps facilitate easy and quick exchange of text messages, photos, videos, and other media, making it convenient to share updates, news, and memories. Users can create groups, share stories, and even make voice or video calls to multiple contacts simultaneously, enhancing the social connectivity and engagement among users.

Social Networking: Real-time chat apps have also been widely used as social networking platforms, connecting people with common interests, hobbies, or communities. These apps provide features such as friend requests, profile creation, status updates, and group chats, allowing users to build and maintain social networks online. Real-time chat apps have facilitated the formation of online communities, where users can discuss topics of mutual interest, share knowledge, and engage in discussions. These apps have also been utilized for

online dating, where users can connect and interact with potential partners in real-time, fostering relationships and social connections.

Business Communication: Real-time chat apps have become an essential tool for business communication, enabling teams to collaborate and communicate in real-time, irrespective of their physical location. These apps provide features such as instant messaging, voice calls, and video calls, facilitating quick and effective communication among team members. Real-time chat apps also offer features like file sharing, document collaboration, and task management, allowing teams to work together on projects and tasks seamlessly. These apps have become particularly important for remote or distributed teams, enabling them to communicate and collaborate in real-time, enhancing productivity and efficiency.

Customer Support: Real-time chat apps have been widely adopted in the customer support domain, providing businesses with a convenient and efficient way to interact with their customers. These apps offer features such as live chat, chatbot integration, and multimedia support, allowing businesses to provide instant assistance and support to their customers. Real-time chat apps enable customers to interact with businesses in real-time, seek assistance, and resolve their queries or issues quickly. These apps have also been utilized for sales and marketing purposes, where businesses can engage with potential customers, provide product information, and offer personalized recommendations.

Education and E-Learning: Real-time chat apps have been increasingly utilized in the education and e-learning domain, providing students and teachers with a platform for real-time communication and collaboration. These apps offer features such as group chats, multimedia sharing, and screen sharing, enabling teachers and students to interact and collaborate in real-time. Real-time chat apps facilitate discussions, debates, and knowledge sharing among students, enhancing the learning experience. Teachers can use these apps to provide real-time feedback, address queries, and conduct virtual classes or webinars. These apps

have become particularly important during the COVID-19 pandemic, where remote learning has become the norm, and real-time chat apps have played a crucial role in bridging the communication gap between students and teachers

Event Management: Real-time chat apps have been used in event management to enhance communication and engagement among event attendees.

These apps provide a platform for event organizers to communicate with attendees, share event updates, and send important notifications in real-time. Attendees can use these apps to connect with each other, share feedback, and interact with event organizers. Real-time chat apps enable event organizers to send instant announcements, conduct polls, and gather feedback from attendees, enhancing the overall experience and engagement of the event.

Gaming and Entertainment: Real-time chat apps have found applications in the gaming and entertainment industry, providing a platform for players to communicate and collaborate during online gaming sessions. These apps offer features such as in-game messaging, voice chats, and social networking, enabling players to interact in real-time, strategize, and coordinate their actions. Real-time chat apps also allow players to form gaming communities, share tips, and organize tournaments or competitions. These apps have also been used in the entertainment industry for live streaming events, where users can interact with content creators, send real-time comments, and engage in discussions, enhancing the overall entertainment experience.

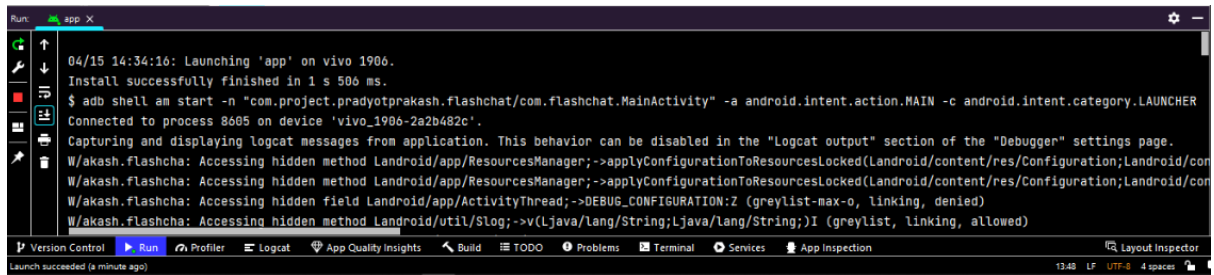
Health Care: Real-time chat apps have been utilized in the health care industry, providing a platform for communication and collaboration among health care professionals and patients. These apps offer features such as secure messaging, video consultations, and appointment scheduling, enabling health care providers to communicate with patients in real-time, provide medical advice, and monitor patient progress. Real-time chat apps have also been used for remote patient

monitoring, where patients can share their health data in real-time, and health care providers can provide real-time feedback and recommendations. These apps have been particularly useful during the COVID-19 pandemic, where telemedicine has become essential for providing medical care remotely.

Emergency Response: Real-time chat apps have been used in emergency response scenarios, providing a platform for communication and coordination among emergency responders during critical situations. These apps offer features such as real-time messaging, location sharing, and multimedia sharing, enabling emergency responders to communicate and collaborate effectively in real-time. Real-time chat apps allow emergency responders to share important information, coordinate rescue efforts, and prioritize response actions. These apps have been utilized during natural disasters, emergencies, and crisis situations, where real-time communication and coordination are crucial for saving lives and managing resources effectively.

Community Building: Real-time chat apps have been used for community building, providing a platform for like-minded individuals to connect, interact, and collaborate on common interests or causes. These apps offer features such as group chats, multimedia sharing, and event organizing, enabling community members to communicate in real-time, share ideas, and work together towards common goals. Real-time chat apps have been used for activism, advocacy, and social change, where communities can organize events, share updates, and mobilize support. These apps have also been utilized for professional networking,

THE AREAS WHERE THIS SOLUTION CAN BE APPLIED:



```
Run: app X
04/15 14:34:16: Launching 'app' on vivo 1906.
Install successfully finished in 1 s 506 ms.
$ adb shell am start -n "com.project.pradyotprakash.flashchat/com.flashchat.MainActivity" -a android.intent.action.MAIN -c android.intent.category.LAUNCHER
Connected to process 8605 on device 'vivo_1906-2a2b482c'.
Capturing and displaying logcat messages from application. This behavior can be disabled in the "Logcat output" section of the "Debugger" settings page.
W/akash.flashcha: Accessing hidden method Landroid/app/ResourcesManager;-->applyConfigurationToResourcesLocked(Landroid/content/res/Configuration;Landroid/con
W/akash.flashcha: Accessing hidden field Landroid/app/ActivityThread;-->DEBUG_CONFIGURATION:Z (greylist-max-o, linking, denied)
W/akash.flashcha: Accessing hidden method Landroid/util/Slog;-->v(Ljava/lang/String;Ljava/lang/String;)I (greylist, linking, allowed)
```



6 CONCLUSION:

Real-time chat apps developed in Android Studio have revolutionized the way we communicate and collaborate in various domains. These apps have become an essential tool in personal communication, social networking, business communication, customer support, education, event management, gaming, health care, emergency response, and community building, among others. The advantages of real-time chat apps include improved communication, enhanced collaboration, increased productivity, and enhanced engagement. However, there are also potential disadvantages, such as security risks, privacy concerns, and potential for misuse.

One of the key conclusions is that real-time chat apps have greatly improved communication by providing instant and convenient communication channels. Users can send and receive messages, make voice and video calls, share files, and send multimedia in real-time, allowing for quick and efficient communication. This has led to increased responsiveness and improved collaboration among individuals and teams, enabling them to work together more effectively.

Another conclusion is that real-time chat apps have greatly enhanced collaboration among individuals and groups. These apps enable users to communicate and collaborate on tasks, projects, or events in real-time, allowing for real-time updates, feedback, and coordination. This has led to increased productivity, as users can work together more efficiently, share ideas, and make decisions in real-time.

Real-time chat apps have also proven to be valuable in various industries, such as customer support, education, event management, and health care. In customer support, real-time chat apps allow businesses to provide instant support to their customers, resolving issues quickly and efficiently. In education, these apps have

been utilized for remote learning, facilitating real-time interaction and collaboration among students and teachers. In event management, real-time chat apps enable organizers to communicate with attendees, share updates, and gather feedback in real-time. In health care, these apps have been used for telemedicine, allowing health care providers to communicate with patients in real-time, monitor their health, and provide timely medical advice.

However, it is important to acknowledge the potential disadvantages of real-time chat apps. Security risks, such as data breaches, hacking, and unauthorized access, can pose a threat to the privacy and confidentiality of user data. Privacy concerns may arise due to the collection and storage of personal information, as well as potential misuse of data by third parties. Additionally, the potential for misuse of real-time chat apps, such as cyberbullying, harassment, and spreading misinformation, is a concern that needs to be addressed.

In conclusion, real-time chat apps developed in Android Studio have unlocked the potential for communication and collaboration in various domains. These apps have greatly improved communication, enhanced collaboration, and increased productivity in personal, social, and professional settings. However, it is important to be mindful of the potential disadvantages and risks associated with real-time chat apps, and take appropriate measures to address them. With proper implementation and management, real-time chat apps can continue to be a valuable tool for enhancing communication and collaboration in the digital era.

7 FUTURE SCOPE

Real-time chat apps developed in Android Studio have become an integral part of our daily lives, transforming the way we communicate and collaborate. As technology continues to evolve, there are numerous exciting possibilities and potential future scope for real-time chat apps using Android Studio. In this section, we will explore some of the potential future directions and advancements for real-time chat apps, spanning across various domains and industries.

Enhanced User Experience: With advancements in user interface (UI) and user experience (UX) design, real-time chat apps can continue to provide more intuitive, engaging, and seamless experiences for users. The use of animations, gestures, and other interactive elements can enhance the overall user experience, making the app more user-friendly and enjoyable to use. Customization options, such as themes, stickers, emojis, and avatars, can also add personalization to the chat app, allowing users to express themselves in unique ways.

Integration with Artificial Intelligence (AI): The integration of AI technologies, such as natural language processing (NLP), machine learning (ML), and chatbots, can open up new possibilities for real-time chat apps. AI-powered chatbots can provide automated responses, suggestions, and recommendations to users, improving customer support, providing personalized content, and automating repetitive tasks. NLP can enable advanced language processing capabilities, such as sentiment analysis, language translation, and voice recognition, making communication more efficient and effective.

Augmented Reality (AR) and Virtual Reality (VR) Integration: The integration of AR and VR technologies into real-time chat apps can revolutionize the way we communicate and collaborate. AR can enable users to share and interact with virtual objects and experiences in real-time, creating immersive and interactive

conversations. VR can enable virtual meeting spaces, where users can gather and collaborate in a virtual environment, regardless of their physical location. This can be particularly beneficial for remote teams, virtual events, and virtual classrooms.

Multi-platform Support: Real-time chat apps can expand their reach by providing support for multiple platforms, such as web browsers, desktop applications, and wearable devices. This can allow users to access the chat app from different devices and platforms, providing flexibility and convenience in communication. Cross-platform compatibility can also enable seamless integration with other applications, services, and APIs, enhancing the functionality and versatility of the chat app.

Enhanced Security and Privacy Features: As the importance of data privacy and security continues to gain attention, future real-time chat apps can focus on providing enhanced security and privacy features. This can include end-to-end encryption, two-factor authentication, user authentication, data encryption, and privacy settings, to ensure the confidentiality and integrity of user data. Regular security audits, vulnerability assessments, and penetration testing can also be conducted to identify and address potential security risks.

Blockchain Integration: The integration of blockchain technology into real-time chat apps can provide added security, transparency, and decentralization. Blockchain can enable secure and verifiable communication, where messages and data are stored in a distributed and immutable ledger. This can prevent data tampering, unauthorized access, and ensure data integrity. Blockchain can also enable the use of cryptocurrency or tokens for transactions within the chat app, providing a secure and transparent way for users to exchange value.

Collaborative Features: Real-time chat apps can further enhance collaboration among users by adding collaborative features, such as shared document editing,

project management, and task tracking. This can enable users to work together in real-time, making updates, assigning tasks, and tracking progress within the chat app itself. This can streamline communication and collaboration, reducing the need for multiple tools and platforms.

Integration of real-time chat apps with Internet of Things (IoT) devices can open up new possibilities for communication and control. Chat apps can be integrated with smart home devices, wearables, and other IoT devices, allowing users to control and monitor their devices through the chat app. For example, users can send voice commands or receive notifications about the status of their smart home devices, track their fitness activities through wearables, and even control IoT devices in their workplace or public spaces, such as smart lights, thermostats, and security systems, all within the chat app.

Gamification elements can be incorporated into real-time chat apps to make the user experience more engaging and interactive. This can include adding game-like features, such as points, rewards, leaderboards, and challenges, to encourage users to actively participate in conversations and achieve goals. Gamification can foster a sense of competition, engagement, and community among users, making the chat app more fun and enjoyable to use.

Monetization Opportunities: Real-time chat apps can explore various monetization opportunities to generate revenue. This can include offering premium features, subscription plans, or in-app purchases for additional functionalities or customization options. Advertising and sponsored content can also be integrated into the chat app, providing a source of revenue. Additionally, partnerships with businesses, e-commerce integration, and affiliate marketing can also be explored to generate revenue through the chat app.

Localization and Globalization: Real-time chat apps can expand their user base by providing localization and globalization features. Localization can involve

translating the chat app into different languages, adapting to regional cultural norms, and providing localized content and features. Globalization can involve expanding the chat app's availability to different regions, countries, and markets, and adapting to local regulations and requirements. This can help in reaching a wider audience and catering to the unique needs of different regions.

Social Impact: Real-time chat apps can have a significant social impact by facilitating communication and collaboration in various domains. For example, chat apps can be used for educational purposes, connecting students and teachers in remote areas, and facilitating online learning. Chat apps can also be used for healthcare purposes, connecting doctors and patients for telemedicine, providing mental health support, and facilitating communication among healthcare professionals. Chat apps can also be used for disaster management, emergency communication, and community engagement, helping to address societal challenges and promote social good.

Personal and Professional Networking: Real-time chat apps can provide opportunities for personal and professional networking, allowing users to connect with like-minded individuals, professionals, and communities of interest. Chat apps can provide features such as user profiles, search and discovery, group chats, and forums, allowing users to expand their network, share knowledge, and collaborate with others. This can be particularly beneficial for professionals, entrepreneurs, freelancers, and individuals looking to build meaningful connections and collaborations.

Customization and Flexibility: Future real-time chat apps can focus on providing customization options and flexibility to users, allowing them to tailor the app to their individual preferences and needs. This can include features such as themes, color schemes, font sizes, notification settings, and chat settings, providing a personalized experience for each user. Customization options can also include the ability to create custom chat rooms, set privacy settings, and

choose preferred communication modes, making the chat app more adaptable to different communication styles and requirements.

Offline Functionality: While real-time chat apps primarily rely on an internet connection for communication, future chat apps can explore offline functionality to ensure uninterrupted communication even in low or no internet connectivity areas. This can involve features such as offline messaging, offline message syncing, and offline data caching, allowing users to send and receive messages, even when they are offline. Offline functionality can be particularly beneficial in remote areas, where internet connectivity is limited or unreliable, ensuring that users can stay connected and communicate effectively even in challenging network conditions.

Enhanced Security and Privacy: With the increasing concerns around data privacy and security, future real-time chat apps can focus on providing enhanced security features to protect user data and privacy. This can include end-to-end encryption for messages, secure authentication methods, and robust data encryption techniques to prevent unauthorized access to user data. Additionally, chat apps can provide privacy settings that allow users to control their visibility, manage their personal information, and choose who can access their messages and content.

Artificial Intelligence and Chatbots: Artificial intelligence (AI) and chatbots can play a significant role in the future of real-time chat apps. AI-powered chatbots can provide personalized recommendations, automated responses, and intelligent suggestions based on user behavior and preferences. Chatbots can also assist in customer support, providing quick and efficient responses to user queries. AI can also be used for sentiment analysis, language translation, and content moderation, enhancing the user experience and making the chat app more intelligent and intuitive.

Virtual Reality and Augmented Reality Integration: Virtual reality (VR) and augmented reality (AR) technologies can revolutionize the way real-time chat apps are used. VR and AR integration can provide immersive and interactive experiences, allowing users to communicate and interact in virtual or augmented environments. For example, users can have virtual meetings, attend virtual events, or share augmented reality content within the chat app. VR and AR integration can add a new dimension to communication, making it more engaging, interactive, and visually appealing.

Cross-Platform Compatibility: Future real-time chat apps can focus on providing cross-platform compatibility, allowing users to communicate seamlessly across different devices and platforms. This can include compatibility with various operating systems, such as Android, iOS, Windows, and macOS, as well as integration with web browsers, smart TVs, and other smart devices. Cross-platform compatibility can ensure that users can communicate with each other, regardless of the devices or platforms they are using, enhancing the convenience and accessibility of the chat app.

Continuous Innovation and Updates: To stay relevant and competitive in the fast-paced world of technology, future real-time chat apps need to focus on continuous innovation and updates. This can involve regularly releasing updates with new features, improvements, and bug fixes, based on user feedback and market trends. Innovation can also involve exploring emerging technologies, such as blockchain, quantum computing, and edge computing, to enhance the functionality, security, and performance of the chat app. Continuous innovation and updates can keep the chat app fresh, exciting, and user-friendly, ensuring its success in the long run.

In conclusion, the future scope for real-time chat apps using Android Studio is vast and promising. With advancements in technology, evolving user needs, and changing market trends, real-time chat apps have the potential to transform

the way we communicate, collaborate, and connect with others. From improved user experience, enhanced functionalities, and integration with emerging technologies, to monetization opportunities, social impact, and customization options, the future of real-time chat apps is bright. Developers using Android Studio can leverage these opportunities and create innovative and feature-rich chat apps that cater to the evolving needs of users, opening up new possibilities and driving the future of communication.

8 APPENDIX:

8.1. Source Code:

Manifests:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com. Quick style ">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/quick"
        android:label="Quick style"
        android:roundIcon="@drawable/quick"
        android:supportsRtl="true"
        android:theme="@style/Theme.FlashChat">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Quick style"
            android:theme="@style/Theme.Quickstyle.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.Quickstyle
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.Quickstyle
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.Quick style.nav.Action
import com.Quick style.nav.Destination.AuthenticationOption
import com.Quick style.nav.Destination.Home
import com.Quick style.nav.Destination.Login
import com.Quick style.nav.Destination.Register
import com.Quick style.ui.theme.Quickstyle Theme
import com.Quick style.view.AuthenticationView
import com.Quick style.view.home.HomeView
import com.Quick style.view.login.LoginView
import com.Quick style.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.Quickstyle
```

```
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. Quickstyle.nav
import androidx.navigation.NavHostController
import com.Quickstyle.nav.Destination.Home
import com.Quickstyle.nav.Destination.Login
import com.Quickstyle .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. Quickstyle.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. Quickstyle Theme
@Composable
fun AuthenticationView(register: () -> Unit, login: () -> Unit) {

```

```

Quickstyle Theme {
theme
    Surface(color = MaterialTheme.colors.background) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .fillMaxHeight(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Bottom
        ) {
            Title(title = "⚡ Quick-style ⚡")
            Buttons(title = "Register", onClick = register, backgroundColor =
Color.Red)
            Buttons(title = "Login", onClick = login, backgroundColor =
Color.Green)
        }
    }
}

```

Widgets:

```
package com. Quickstyle.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. Quickstyle.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. Quickstyle.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. Quickstyle.Constants
import com. Quickstyle.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 = TextInputType.Text
),
singleLine = true,
trailingIcon = {
    IconButton(
        onClick = {
            homeViewModel.addMessage()
        }
    ) {
        Icon(
            imageVector = Icons.Default.Send,
            contentDescription = "Send Button"
```

```

        )
    }
}
)
}
}

```

HomeViewModel class:

```

package com. Quickstyle.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. Quickstyle.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.isEmpty()) {
        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. Quickstyle.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. Quickstyle.view.Appbar
import com. Quickstyle.view.Buttons
import com. Quickstyle.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. Quickstyle.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. Quickstyle.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. Quickstyle.view.Appbar
```

```

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

```

RegisterViewModel class:

```
package com. Quickstyle.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
            }
    }
}

```

The Solution Built:

The image displays two screenshots of the Android Studio IDE, showing the development and execution of an Android application named 'FlashChat'.

Top Screenshot: The IDE shows the 'AndroidManifest.xml' file. The manifest includes the following elements:

```
<?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="Quick style"
        android:roundIcon="@drawable/quick"
        android:supportRtl="true"
        android:theme="@style/Theme.FlashChat">

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Quick style"
            android:theme="@style/Theme.Quickstyle.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

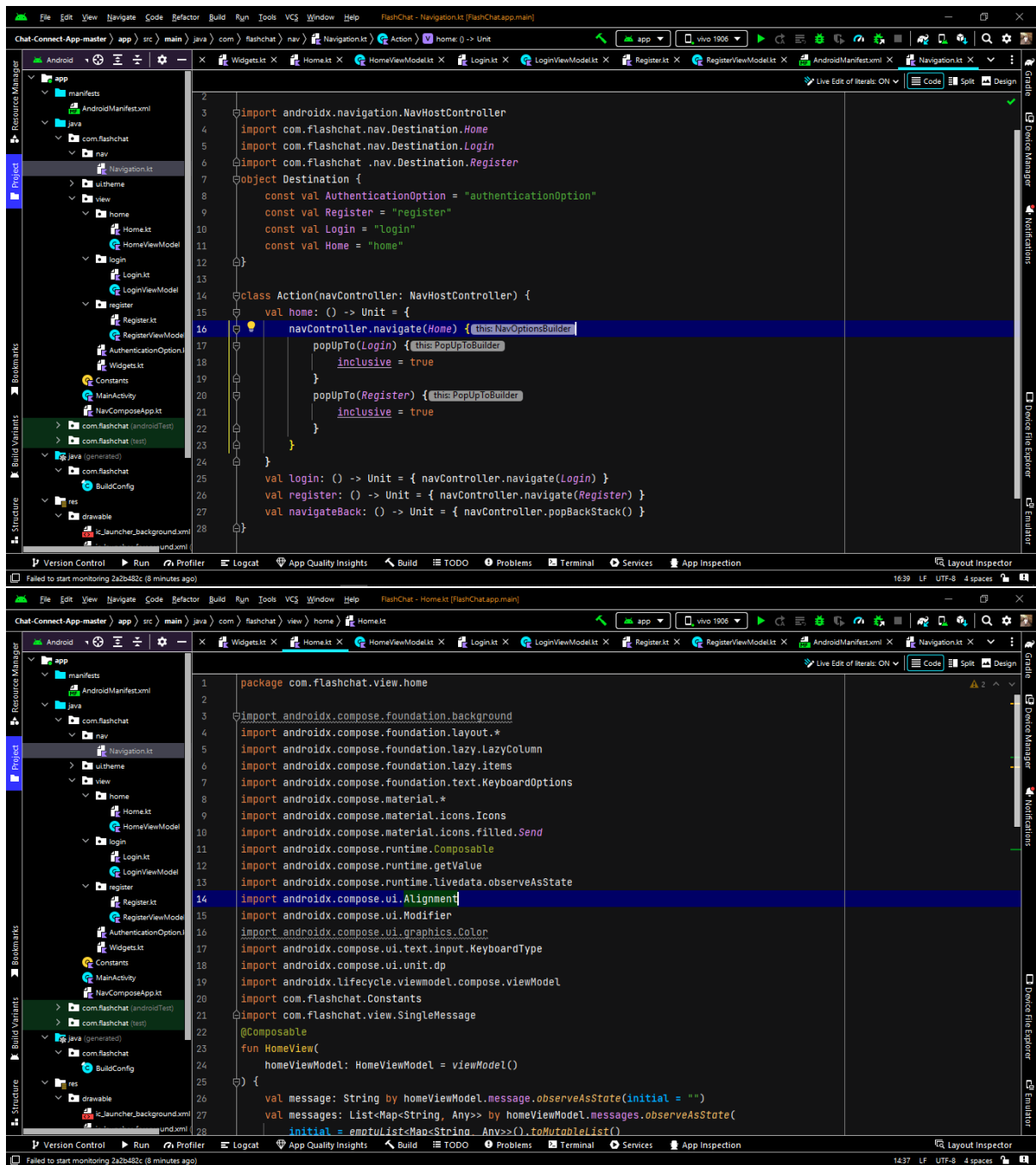
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

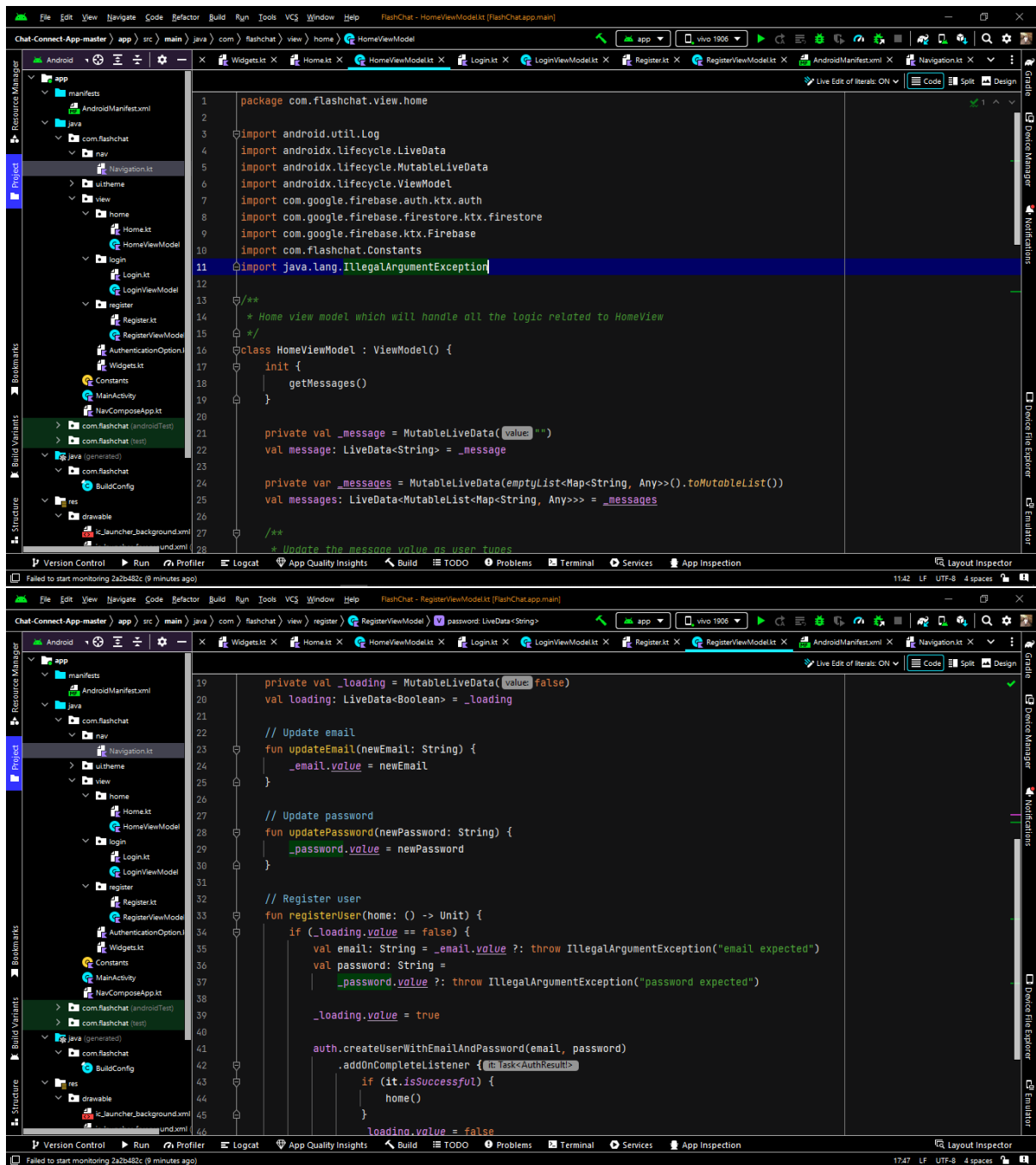
The bottom toolbar shows the 'Run' button (a green play icon) is highlighted.

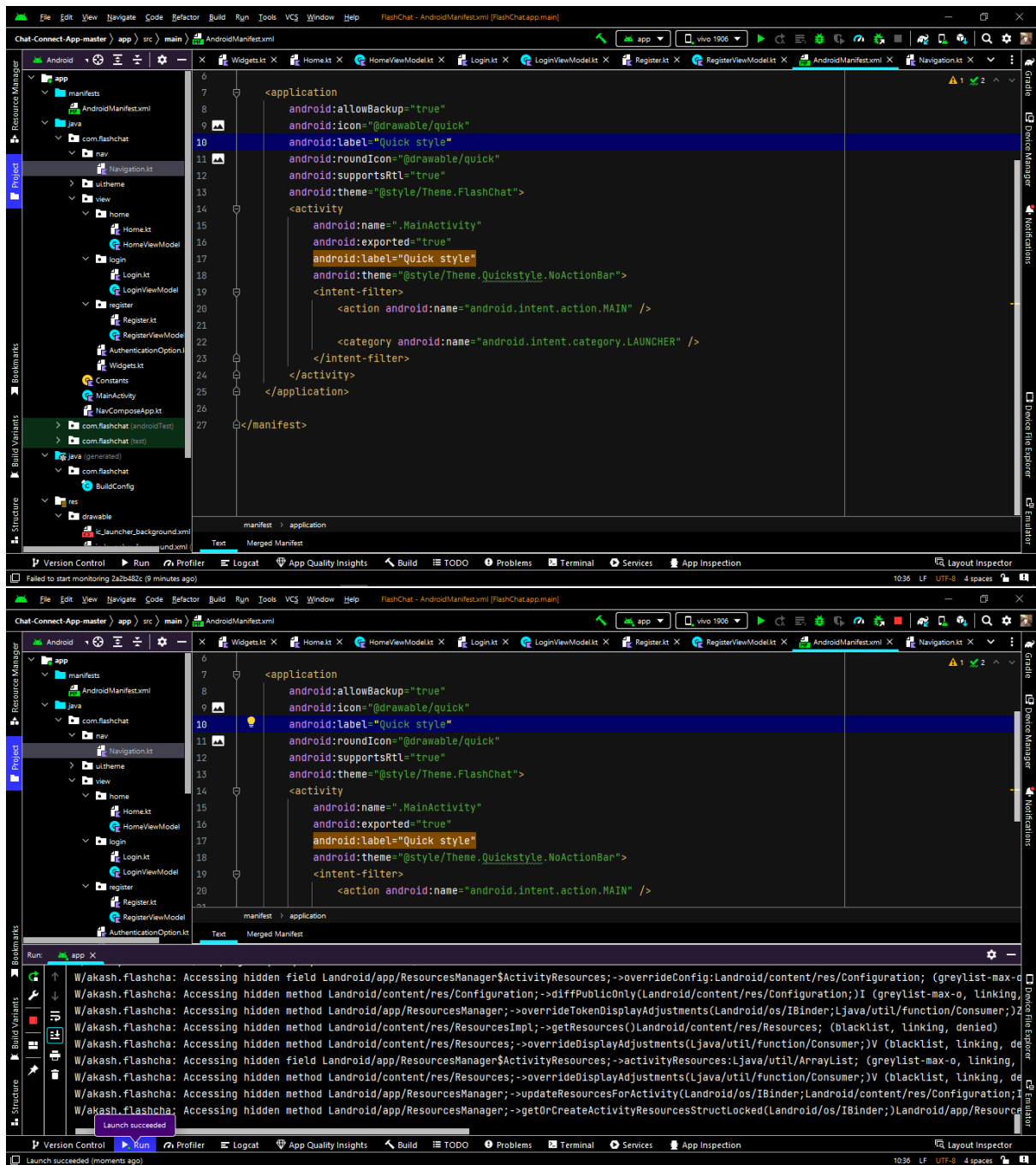
Bottom Screenshot: The IDE shows the same 'AndroidManifest.xml' file, but the 'Run' button is now disabled. The 'Run' tab at the bottom shows the following output:

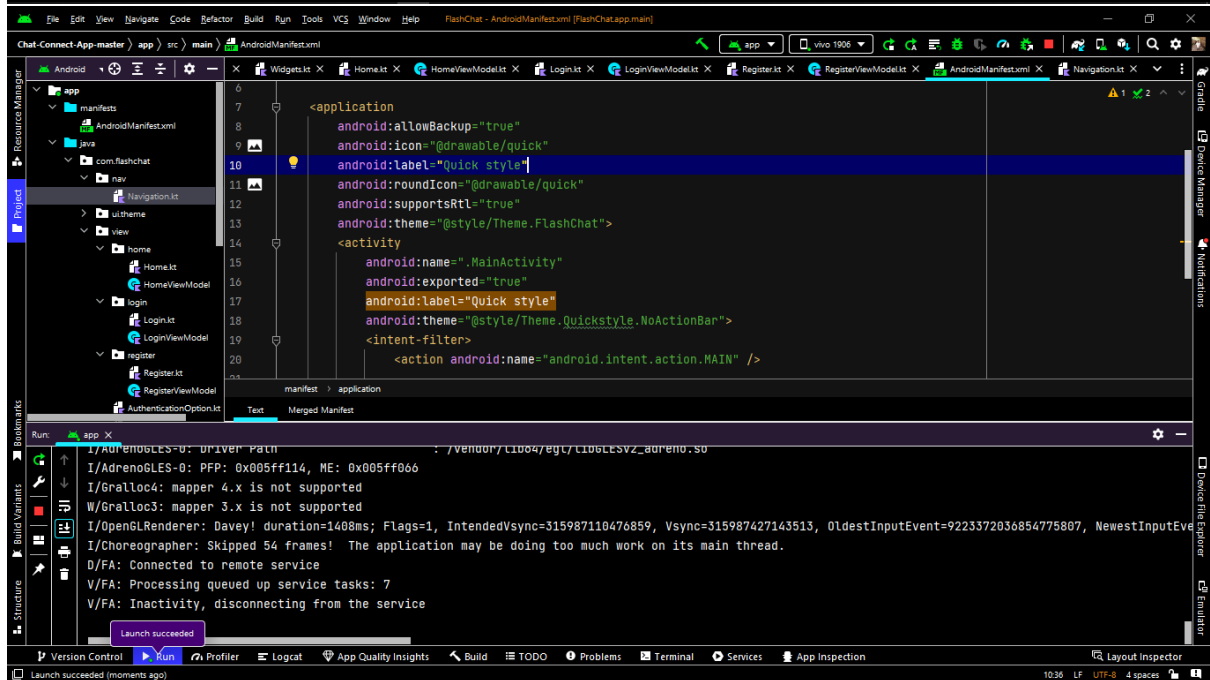
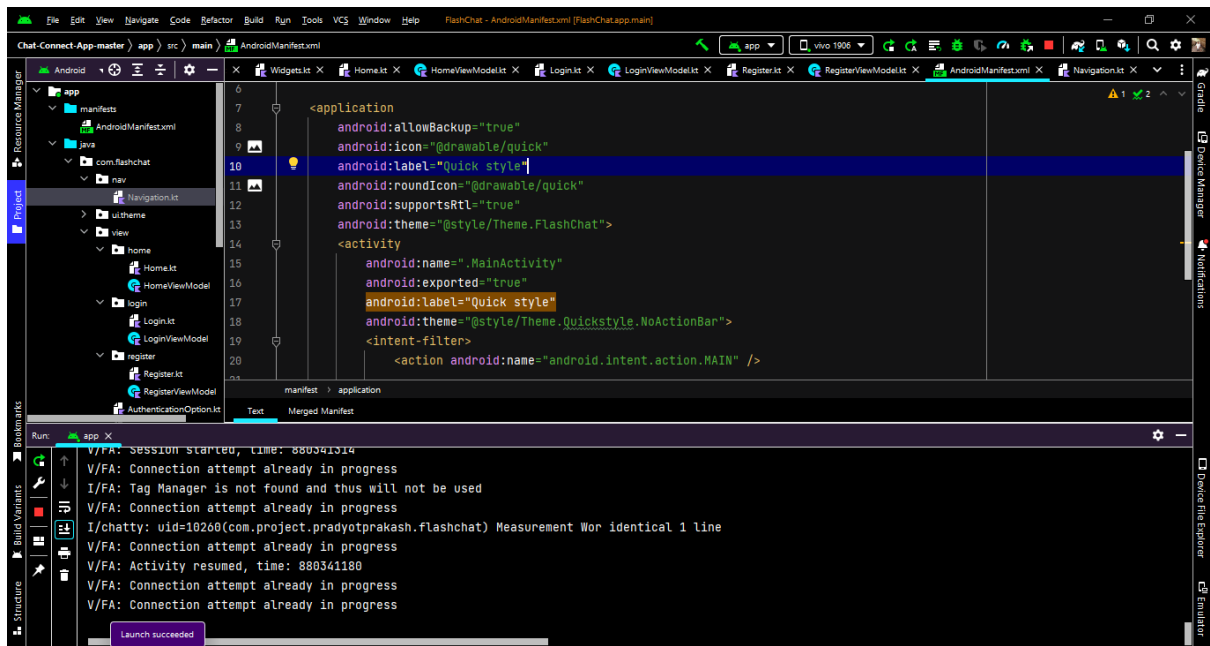
```
04/15 14:34:16: Launching 'app' on vivo 1906.
Install successfully finished in 1 s 586 ms.
$ adb shell am start -n "com.project.pradyotprakash.flashchat.MainActivity" -a android.intent.action.MAIN -c android.intent.category.LAUNCHER
Connected to process 8605 on device 'vivo_1906-2a2b482c'.
Capturing and displaying logcat messages from application. This behavior can be disabled in the "Logcat output" section of the "Debugger" settings page.
W/akash.flashcha: Accessing hidden method Landroid/app/ResourcesManager;->applyConfigurationToResourcesLocked(Landroid/content/res/Configuration;Landroid/con
W/akash.flashcha: Accessing hidden method Landroid/app/ResourcesManager;->applyConfigurationToResourcesLocked(Landroid/content/res/Configuration;Landroid/con
W/akash.flashcha: Accessing hidden field Landroid/app/ActivityThread;:>DEBUG_CONFIGURATION:Z (greylist-max-o, linking, denied)
W/akash.flashcha: Accessing hidden method Landroid/util/Slog;:>v(Ljava/lang/String;Ljava/lang/String;)I (greylist, linking, allowed)
```

The bottom toolbar shows the 'Run' button is now disabled, and the 'Run' tab is active.

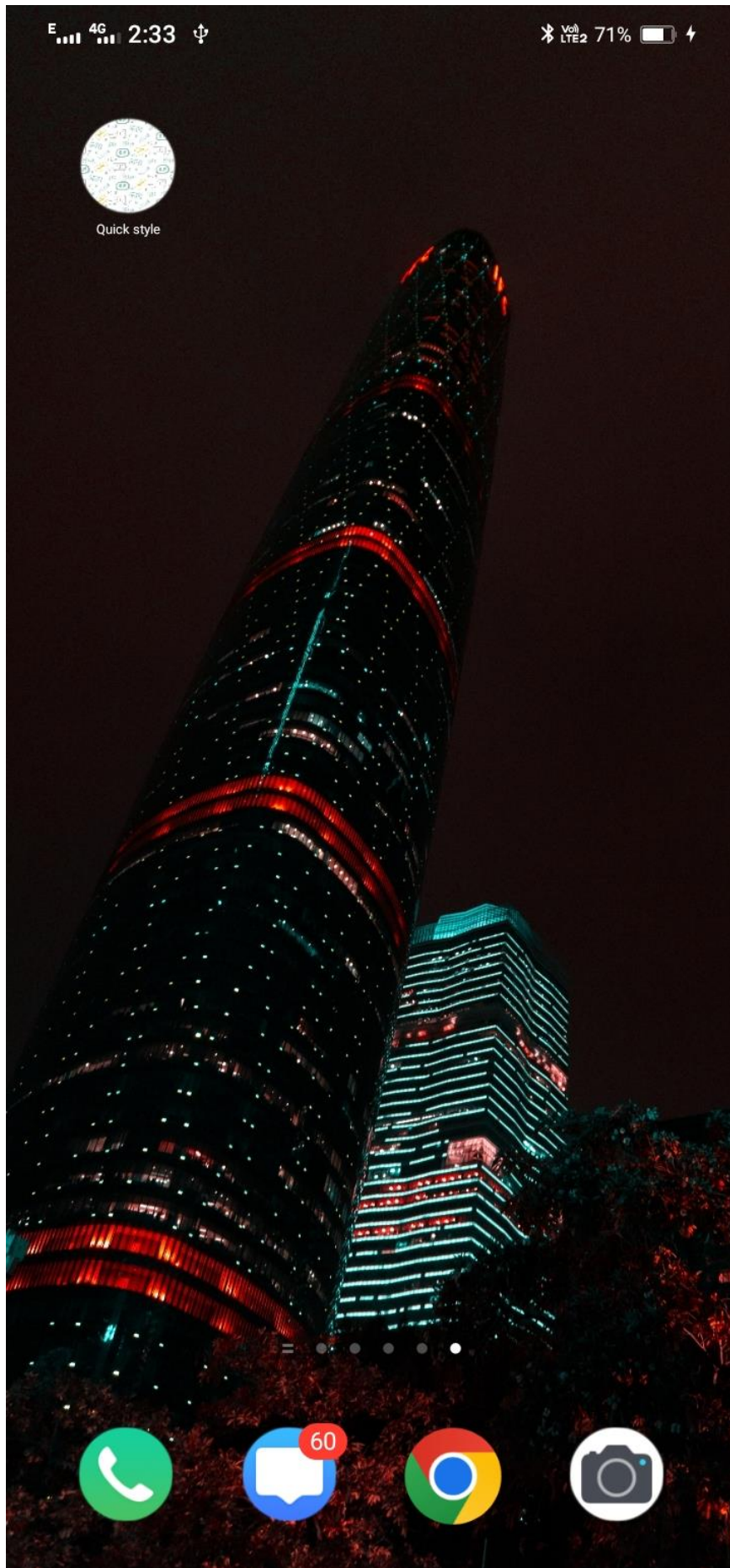








OUTPUT:



⚡ Quick-style ⚡

Register

Login



Register

Email

Password

Register



Login

Email

Password

Login

Hi

Hi

How are you

Type Your Message

>