Friendly Chat

A PROJECT REPORT
for
Major Project (KCA353)
Session (2023-24)

Submitted by

VISHAL SEN

(2200290140182)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of

Mr. Akash Rajak

(Associate Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(March-2024)

DECLARATION

I hereby declare that the work presented in report entitled “Friendly chat” was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University of Institute. I have given due credit to the original authors/ sources for all the words, ideas, diagrams, graphics, computer programs, that are not my original contribution. I have used quotation marks to identify verbatim sentences and give credit to the original authors/sources. I affirm that no portion of my work is plagiarised, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name: Vishal Sen

Roll No.: 2200290140182

CERTIFICATE

Certified that **Vishal Sen 2200290140182** have carried out the project work having "**Friendly Chat**" (**Major Project-KCA353**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the students themselves and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Vishal Sen 2200290140182

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Akash Rajak	**Dr. Arun Tripathi**
(Associate Professor)	**Head**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Groups of Institutions, Ghaziabad**

ABSTRACT

The project is a real-time chatting app built using the Flutter framework, which allows for easy integration of multimedia sharing features such as photo, video, and audio files. The app has a range of functionalities, including one-to-one and group chatting, photo sharing, and multimedia sharing.

The app is powered by a reliable backend that can handle high traffic and messaging volumes. The backend is built using a backend-as-a-service (BaaS) provider like Firebase or a custom-built backend using Node.JS or another backend technology.

The user experience is a major focus of the app, with an intuitive and easy-to-use interface designed to support different user actions. Users can easily navigate through the app to find their friends, create groups, and share multimedia files.

To ensure the security of user data, the app implements end-to-end encryption, user authentication, and secure file transfer protocols. This ensures that user data is protected at all times and prevents unauthorised access to user data.

Testing is a critical component of the project, with the app undergoing both manual and automated testing to identify and fix any bugs or performance issues. This ensures that the app is stable and reliable and provides a seamless user experience.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my thesis supervisor, Mr. Akash Rajak (Associate Professor) for his guidance, help and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions have guided me a lot in completing this project successfully.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions. Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Vishal Sen (2200290140182)

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Our project aims to develop a feature-rich chatting app that offers a seamless and engaging communication experience for users. The app will include essential features such as photo sharing, group chatting, one-to-one chatting, and multimedia sharing. It will be available on both web and mobile platforms, catering to a wide range of users.

The app will provide a user-friendly interface designed to enhance usability and ensure a smooth navigation experience. Users will be able to create accounts and log in securely, using authentication mechanisms such as email/password or social media login. Once logged in, they will have access to a personalised dashboard where they can manage their chats, groups, and multimedia content.

The core functionality of the app will revolve around real-time communication. We will implement a robust real-time communication protocol, such as WebSocket, to enable instant messaging between users. This will allow users to engage in one-to-one conversations as well as participate in group chats with multiple participants. The app will support rich media content, enabling users to share photos, videos, and other multimedia files seamlessly.

To facilitate photo sharing, we will design and implement a secure storage solution. Users will be able to upload and share photos directly from their devices, with options to apply filters, captions, and other editing features. The app will prioritise data security and privacy, ensuring that all media files are stored securely and inaccessible to unauthorised users. Additionally, the app will incorporate push notifications to keep users informed about new messages, group activity, or other relevant events. This will help users stay connected and engaged even when they are not actively using the app. Throughout the development process, we will focus on rigorous testing and debugging to ensure a high-quality product. We will also seek user feedback during beta testing and post-launch to continuously improve the app based on user needs and preferences.

By creating this feature-rich chatting app, we aim to provide a platform where users can connect, communicate, and share moments with ease. Our goal is to deliver an intuitive and secure chatting experience that fosters meaningful connections among users, making the app a go-to choice for their communication needs.

1.2 LITERATURE REVIEW

The rise of messaging apps has revolutionised the way people communicate in today's digital age. These apps have become a central part of daily communication, offering convenient and instantaneous ways to connect with others. This literature review examines key research findings and trends related to messaging apps, focusing on their impact, features, user behaviour, and privacy concerns.

Several studies have explored the impact of messaging apps on communication patterns and social relationships. For instance, a study by Döring and Pöschl (2019) investigated how messaging apps influence the intensity and satisfaction of social relationships. They found that frequent use of messaging apps was associated with stronger social ties and higher relationship satisfaction. Another study by Jin and Park (2019) examined the effects of messaging apps on intergenerational communication, highlighting how these apps facilitate family connections across different age groups.

Features play a crucial role in messaging app design, influencing user satisfaction and adoption. Multimedia sharing is a key feature that enhances communication experiences. Research by Liu and Liang (2019) emphasised the significance of multimedia sharing capabilities in messaging apps, noting that users appreciate the ability to share photos, videos, and audio files. Group chat functionality has also been extensively studied. A study by Hu et al.

(2017) investigated the impact of group chats on social cohesion and found that group chat usage was positively associated with increased social bonding and a sense of belonging.

Privacy concerns have gained attention in the realm of messaging apps. Users are increasingly conscious of their personal data and information security. A study by Xu and Teo (2019) explored privacy concerns and trust in messaging apps, highlighting the importance of transparent privacy policies, secure encryption, and control over data sharing. Researchers have also examined the role of end-to-end encryption in ensuring user privacy and data security. A study by Liao and Fu (2017) emphasised that end-to-end encryption enhances user trust by safeguarding messages from unauthorized access.

The impact of messaging apps on user behaviour has been a subject of interest. Research by Chang and Chuang (2020) examined how messaging apps affect users' multitasking behaviours. They found that messaging app usage leads to increased multitasking, as users tend to engage in various activities simultaneously while communicating through the app. Additionally, studies have explored the impact of messaging apps on individuals' psychological well-being. For instance, a study by Chua and Chang (2019) found that the use of messaging apps can positively influence users' self-esteem and reduce feelings of loneliness.

As messaging apps continue to evolve, their future trends and advancements have also been explored. A study by Kim and Kim (2020) discussed the integration of artificial intelligence (AI) and chatbots within messaging apps, enabling automated responses and personalized experiences. The study highlighted the potential of AI to enhance user interactions and streamline communication processes.

In conclusion, messaging apps have significantly transformed communication patterns and social relationships. Key features such as multimedia sharing and group chat functionality contribute to user satisfaction and social bonding. Privacy concerns and data security are critical considerations in messaging app design, and the implementation of end-to-end encryption plays a crucial role in establishing user trust. Moreover, messaging apps have been found to influence user behavior and psychological well-being positively. The integration of AI and chatbots presents exciting opportunities for future developments in messaging apps.

1.3 PROJECT SCOPE

The scope of our project is to develop a comprehensive chatting app with a range of features including photo sharing, group chatting, one-to-one chatting, and multimedia sharing. The app will target both web and mobile platforms, providing a seamless user experience across devices.

The primary objective of the app is to enable users to connect and communicate effectively. Users will be able to create accounts securely, either through email/password authentication or by logging in via social media accounts. Once logged in, they will have access to a personalized dashboard where they can manage their chats, groups, and multimedia content.

The core functionality of the app will revolve around real-time communication. Users will be able to engage in one-to-one conversations with friends, family, or colleagues, facilitating private and direct communication. Additionally, the app will support group chats, allowing users to create or join groups based on their interests, communities, or organizations. This feature will enable multiple participants to interact, share information, and collaborate in real time.

Another crucial aspect of the app is multimedia sharing. Users will have the ability to share photos, videos, and other multimedia content seamlessly within the app. They will be able to upload media files directly from their devices, apply filters or captions if desired, and share them with selected individuals or groups. Robust and secure storage mechanisms will be implemented to ensure the privacy and integrity of shared multimedia content.

To enhance user engagement and keep them informed, the app will incorporate push notifications. Users will receive notifications for new messages, group activities, or other relevant events even when the app is not actively in use. This feature will help users stay connected and respond promptly to important conversations or updates.

Throughout the development process, emphasis will be placed on ensuring data security, privacy, and a smooth user experience. Rigorous testing and debugging will be conducted to identify and resolve any issues or bugs that may arise. User feedback will be actively sought and considered to continuously improve the app and enhance user satisfaction.

1.4 HARDWARE / SOFTWARE USED IN PROJECT

1.4.1 SOFTWARE REQUIREMENTS

- Operating System – Windows 10/11 or Ubuntu 18.04 or above.
- Code Editor – Microsoft Visual Studio Code
- Flutter: for designing UI
- Dart: for programming language
- Firebase Firestore: for online database

1.4.2 HARDWARE REQUIREMENTS

- Processor – Intel i5 7th generation or higher
- RAM – Minimum 4 GB, recommended 8 GB
- Disk Space – Minimum 10 GB of free disk space

CHAPTER 2

FEASIBILITY STUDY

2.1 TECHNICAL FEASIBILITY

The technical feasibility study is conducted to evaluate the practicality and viability of implementing the proposed features and functionalities of the chatting app. It assesses the technical requirements, resources, and capabilities necessary for successful development and deployment. Here are the key aspects to consider:

- **Platform Compatibility:** Assess the compatibility of the chosen programming languages and frameworks with the targeted platforms (web and mobile). Ensure that the selected technologies can support real-time communication, multimedia sharing, and other desired features on both platforms. Consider the availability of cross-platform frameworks like React Native or Flutter to streamline development for multiple platforms.

- **Infrastructure and Scalability:** Evaluate the required infrastructure to support the anticipated user base. Determine if the necessary servers, databases, and network resources are available or can be provisioned. Consider scalability options such as cloud hosting to accommodate increased user demand and ensure the app's performance remains consistent.

- **Authentication and Security:** Evaluate the technical feasibility of implementing secure user authentication mechanisms. Consider the availability of libraries, frameworks, or services that provide robust authentication and encryption protocols to safeguard user data and prevent unauthorised access. Ensure compliance with relevant security standards and best practices.

- **Real-time Communication:** Assess the technical feasibility of integrating real-time communication protocols, such as WebSockets or MQTT, to facilitate instant messaging between users. Determine if the chosen technology can handle concurrent connections, message synchronisation, and real-time updates effectively.

2.2 OPERATIONAL FEASIBILITY

Operational feasibility refers to assessing whether a proposed project, such as the development of a chatting app with various features, is practical and achievable from an operational standpoint. It involves evaluating the project's impact on existing operations, resources, and processes. Here are some key considerations for operational feasibility:

- **User Acceptance:** Assess the willingness of potential users to adopt and utilise the chatting app. Conduct surveys, interviews, or user research to understand user preferences, needs, and expectations. Evaluate whether the app's features align with user requirements and if it offers a user experience that is intuitive and valuable.

- **Organizational Impact:** Determine the impact of introducing the chatting app on existing organizational structures, processes, and workflows. Assess whether the organization has the capacity to support and maintain the app. Consider the need for additional resources, such as personnel or training, to effectively operate and manage the app.

- **Integration with Existing Systems:** Evaluate the compatibility of the chatting app with any existing systems or platforms used within the organization. Determine if the app can integrate smoothly with other software or databases to ensure seamless data flow and information sharing.

- **Scalability and Performance:** Consider the ability of the app to handle increased user demand and growth over time. Assess whether the infrastructure and resources can scale effectively to accommodate a larger user base without compromising performance. Plan for system upgrades or enhancements if necessary.

2.3 BEHAVIORAL FEASIBILITY

Behavioral feasibility, also known as social or human feasibility, focuses on assessing whether the proposed chatting app aligns with the behavioral and social aspects of its intended users. It involves understanding user behaviors, attitudes, and preferences to ensure the app meets their needs and is embraced by the target audience. Here are some key considerations for behavioral feasibility:

- **User Behavior and Habits:** Analyze the existing behaviors and habits of the target users regarding communication and messaging apps. Understand how they currently interact, share information, and engage with others. Identify any specific preferences or patterns in their communication habits.

- **User Acceptance and Adoption:** Determine the likelihood of user acceptance and adoption of the chatting app. Consider factors such as the app's user interface, ease of use, intuitiveness, and compatibility with existing user behaviors. Assess whether the app offers features and functionalities that align with users' communication needs and preferences.

- **User Experience (UX):** Evaluate the app's user experience design to ensure it provides a seamless and enjoyable interaction for users. Consider factors such as navigation, responsiveness, visual appeal, and overall usability. Conduct user testing and gather feedback to refine the app's UX design and enhance user satisfaction.

- **Privacy and Security:** Address user concerns regarding privacy and security. Ensure the app implements robust security measures to protect user data, conversations, and shared content. Clearly communicate the app's privacy policies and establish trust with users by prioritizing their data protection.

- **Feedback and Iteration:** Establish channels for users to provide feedback and suggestions. Actively seek user input during the development and post-launch phases to understand their needs, address any issues, and iterate on the app's features and functionalities. Incorporate user feedback into ongoing app improvements and updates.

2.4 ECONOMICAL FEASIBILITY

Economical feasibility, also known as financial feasibility, assesses the financial viability and potential return on investment of a proposed project, such as the development of a chatting app. It involves evaluating the costs associated with the project and determining if the benefits outweigh the investment. Here are some key considerations for economical feasibility:

- **Cost Estimation:** Evaluate the costs involved in developing and deploying the chatting app. Consider factors such as software development, design, infrastructure setup, hosting, licensing fees, and any other associated expenses. Estimate both one-time costs and ongoing operational costs.

- **Benefit Analysis:** Identify and quantify the potential benefits that the chatting app can bring to the organization or target users. These benefits can include increased productivity, improved communication, cost savings, or enhanced user satisfaction. Determine how these benefits translate into financial gains or savings.

- **Return on Investment (ROI):** Calculate the expected ROI by comparing the projected benefits with the estimated costs. Assess the payback period and determine if the financial returns justify the initial investment. Consider the timeframe for achieving profitability and the long-term financial sustainability of the project.

- **Revenue Generation:** Explore potential revenue streams that can be generated through the chatting app. These can include in-app purchases, premium features or subscriptions, advertising, or partnerships. Evaluate the market potential and competition to estimate revenue generation and its impact on the overall financial feasibility.

- **Cost Reduction:** Assess the potential cost reduction that the app can bring to the organization or users. For example, if the app streamlines communication and collaboration processes, it may lead to savings in time, resources, or operational costs. Quantify these cost reductions to determine their impact on the overall feasibility.

- **Risk Assessment:** Identify potential financial risks and uncertainties associated with the project. Evaluate factors such as market competition, changing user preferences, evolving technology, and regulatory compliance. Mitigate risks through careful planning, market analysis, and contingency strategies.

CHAPTER 3

DATABASE DESIGN

3.1 WATERFALL MODEL

The waterfall model is a well-known structured methodology for software development. The whole process of system development is divided into distinct phases. The model has been introduced in 1970s. Every phase has a unique output. It was the first SDLC model to be used widely. So that, sometimes it is referred to Waterfall by SDLC. The waterfall model is used when the system requirements are well known, technology is understood, and the system is a new version of an existing
product (Dennis, Wixom and Roth, 2012).

Mainly there are six phases in Waterfall model. If there is a problem faced in any phase of cycle, the system goes to the previous phase. The phases of Waterfall method are.

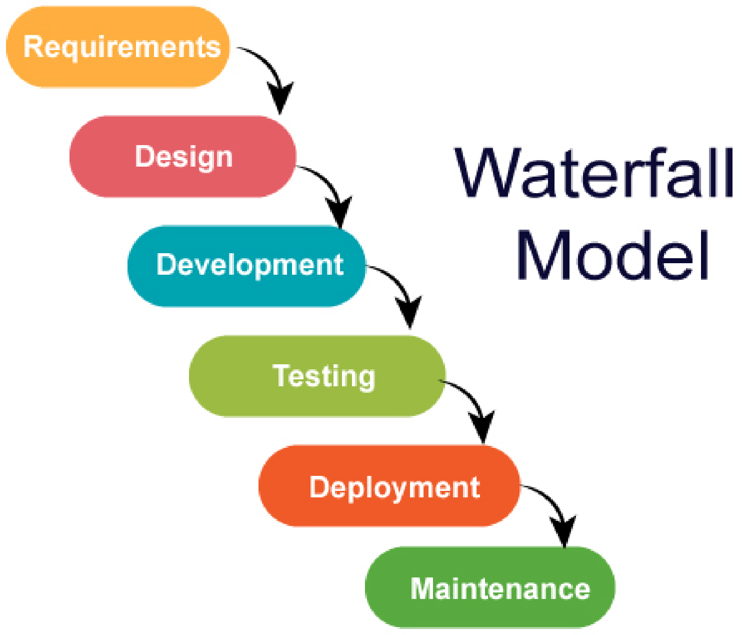


Figure 1 : Waterfall Model

3.2 ER DIAGRAM

An Entity-Relationship (ER) diagram can be utilized in the development of a chatting app to visualize the relationships between various entities and their attributes. Here's how an ER diagram can be applied.

- **User Entity:** The ER diagram can include a "User" entity, representing the users of the chatting app. It can have attributes such as UserID, Username, Password, Email, and Profile Picture. This entity captures the basic information related to the app's users.

- **Chat Entity:** The "Chat" entity can be included to represent individual conversations or group chats. It may have attributes such as ChatID, Title, and CreationDate. This entity helps establish the relationship between users and their conversations.

- **Message Entity:** The "Message" entity can be used to capture the messages exchanged within a chat. It can have attributes such as MessageID, Content, SenderID, and Timestamp. This entity establishes the relationship between chats and the messages exchanged within them.

- **Group Entity:** If the app includes group chatting functionality, a "Group" entity can be added. It can have attributes such as GroupID, Name, Description, and CreationDate. This entity helps represent and manage the groups within the app.

- **Media Entity:** If multimedia sharing is a feature of the app, a "Media" entity can be included. It may have attributes like MediaID, Type, Filename, and UploadDate. This entity enables the management and tracking of shared multimedia content.

- **Relationships:** Establish relationships between entities using appropriate cardinalities. For example, a user can participate in multiple chats, so the relationship between User and Chat would be one-to-many. Similarly, messages belong to a specific chat, indicating a one-to-many relationship between Chat and Message.

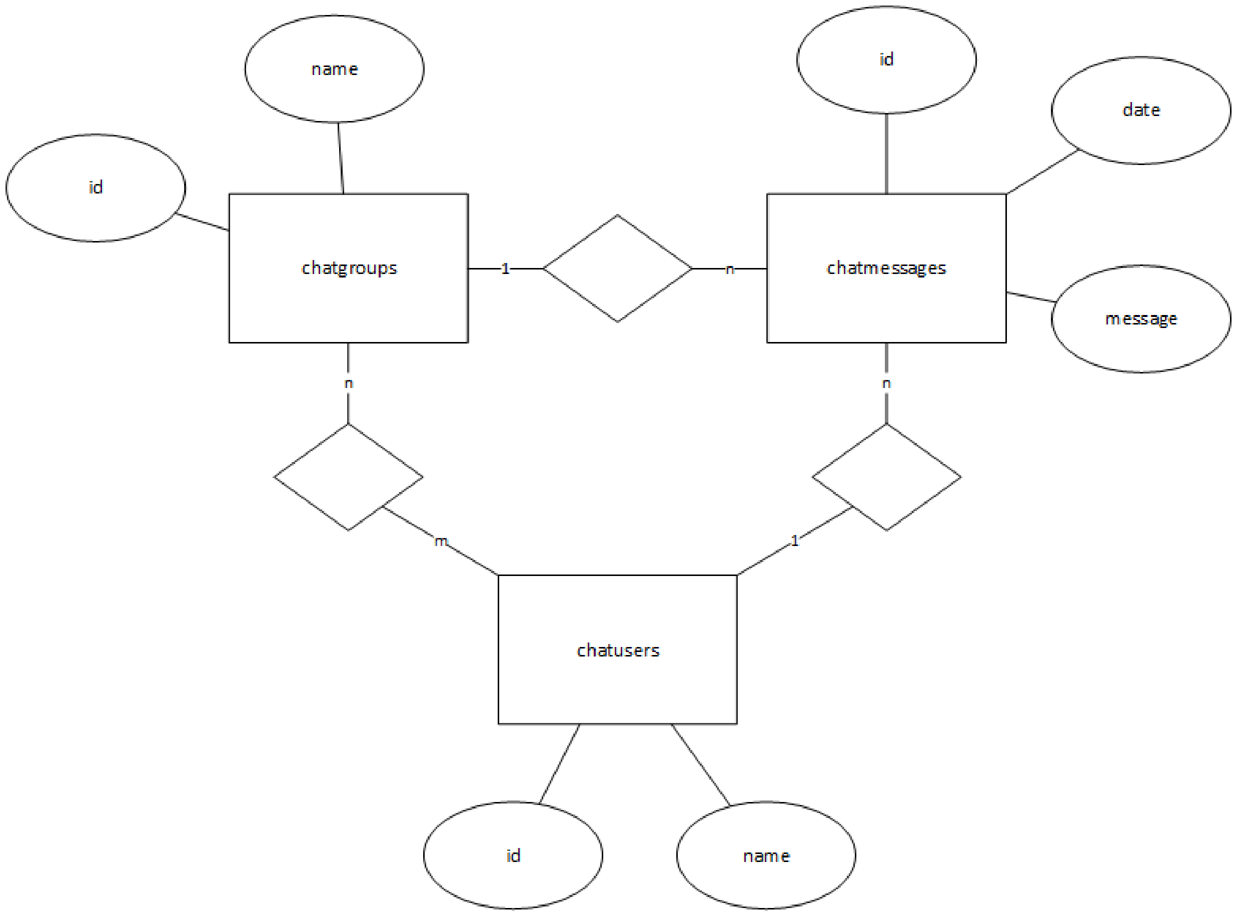


Figure 2 : ER Diagram

3.3 USE CASE DIAGRAM

Use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purposes of a use case diagram given below:

- It gathers the system's needs.
- It depicts the external view of the system.
- It recognises the internal as well as external factors that influence the system.
- It represents the interaction between the actors

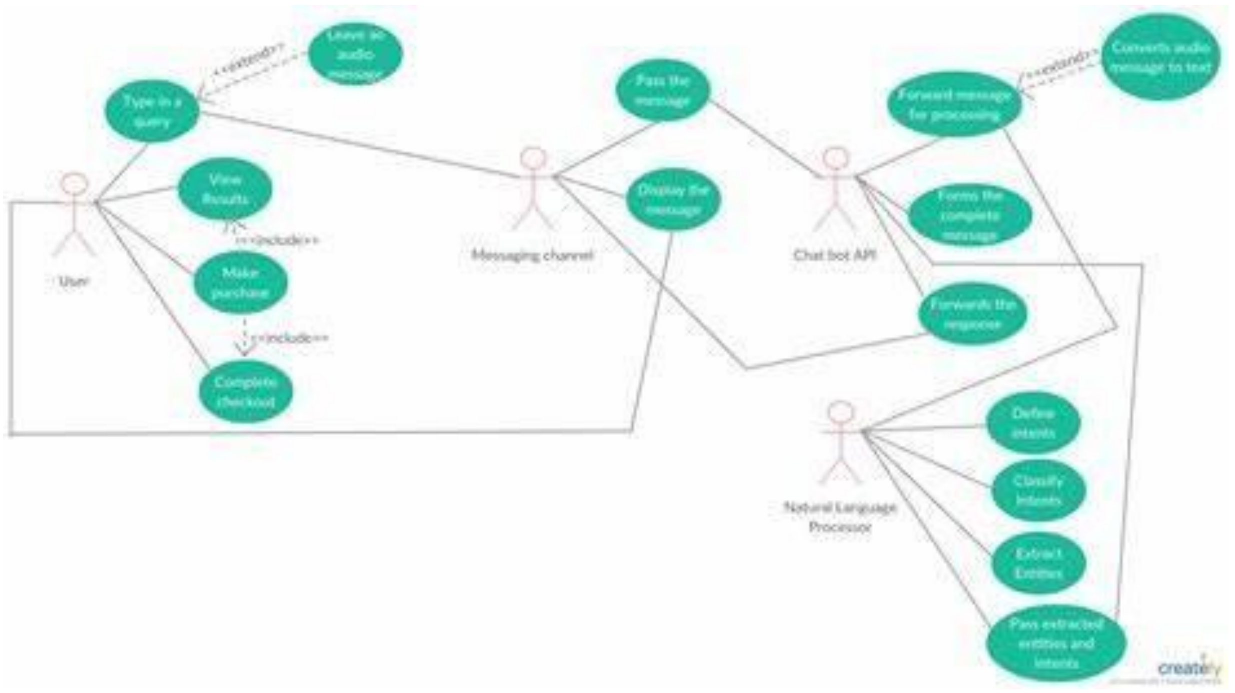


Figure 3 : Use Case Diagram

3.4 ACTIVITY DIAGRAM

An Activity Diagram can be used in the development of a chatting app to depict the flow of activities and behaviours within the system. Here's how an Activity Diagram can be applied.

- **User Registration:** The Activity Diagram can start with the "User Registration" activity, depicting the process of a user creating an account. It would include activities such as entering registration details, validating input, and creating a new user profile.

- **Login and Authentication:** The diagram can show the activities related to user login and authentication. This would involve activities like entering login credentials, validating them, and granting access to the app upon successful authentication.

- **Chatting Activities:** The core activities of the chatting app can be represented, including creating a new chat, joining an existing chat, and sending messages. These activities can be visualized as separate swimlanes or branches, showing the interaction between multiple users or participants within the chat.

- **Multimedia Sharing:** If the app supports multimedia sharing, the diagram can depict the activities involved in uploading and sharing photos, videos, or other media files. It would include activities like selecting a file, uploading it, and notifying other participants of the shared media.

- **Group Management:** If the app includes group chatting functionality, the Activity Diagram can illustrate activities related to group creation, management, and member invitations. It would involve activities like creating a new group, adding or removing members, and moderating group conversations.

- **Notification System:** The diagram can show activities related to the notification system, including activities like sending push notifications for new messages or updates, and displaying notifications to the users.

- **Logout and Session Management:** The diagram should include activities related to user logout and session management, ensuring that users can securely end their session and handle scenarios such as session timeouts.

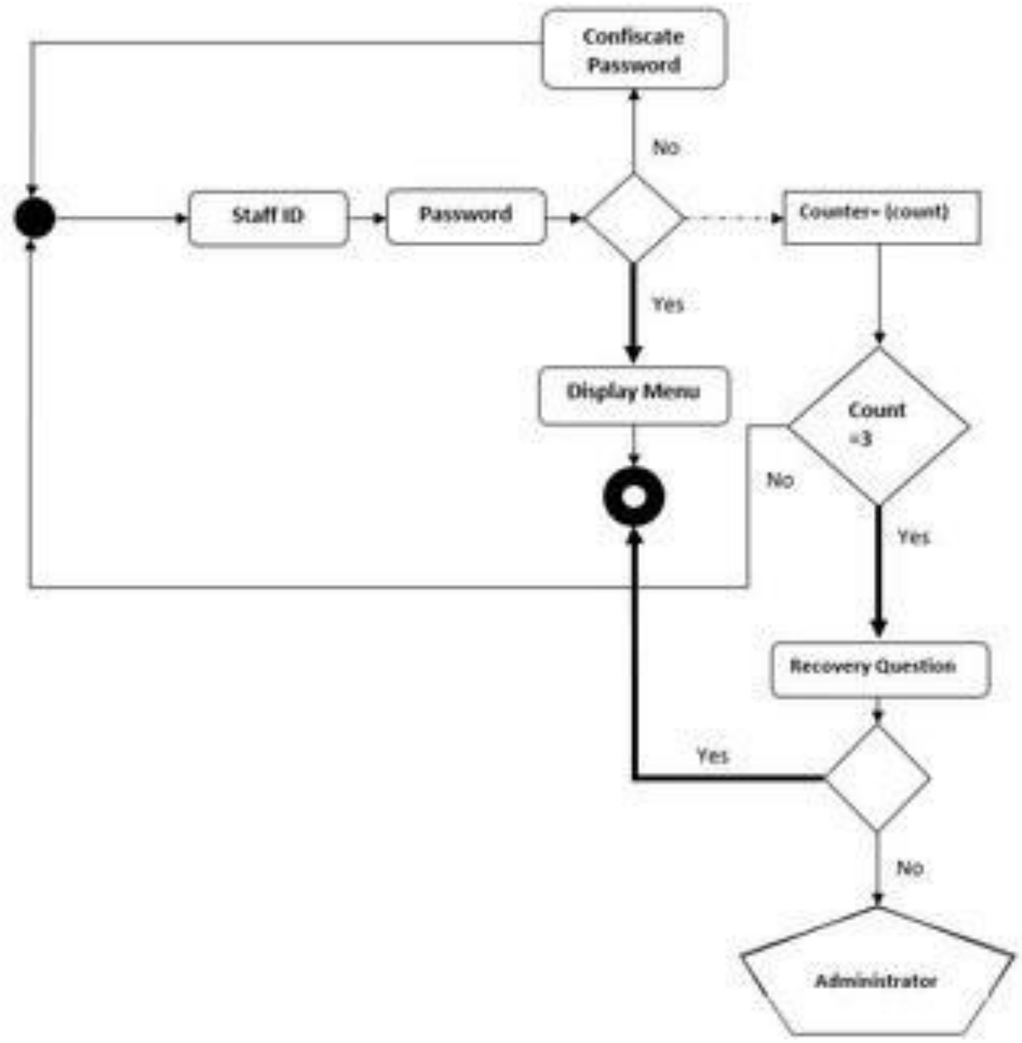


Figure 4 : Activity Diagram

3.5 SEQUENCE DIAGRAM

A Sequence Diagram can be useful in the development of a chatting app to illustrate the chronological sequence of interactions between different entities or components within the system. Here's how a Sequence Diagram can be applied:

- **User Registration:** The Sequence Diagram can depict the sequence of interactions between the user interface and the backend components during the user registration process. It would show the steps involved, such as the user entering registration details, the system validating the input, and the creation of a new user profile.

- **Login and Authentication:** The diagram can illustrate the sequence of interactions between the user, the login interface, and the authentication system. It would show how the user enters login credentials, the system verifies them, and grants access to the app upon successful authentication.

- **One-to-One Chat:** The Sequence Diagram can represent the sequence of interactions between two users engaging in a one-to-one chat. It would show the exchange of messages between the sender and receiver, indicating the order and timing of message delivery.

- **Group Chat:** If the app supports group chatting, the diagram can depict the interactions between multiple users within a group chat. It would show how messages are sent, received, and distributed to all members of the group.

- **Multimedia Sharing:** The Sequence Diagram can illustrate the interactions involved in sharing multimedia content, such as photos or videos. It would show the steps of selecting a file, uploading it to the server, and notifying other users about the shared media.
- **Notification System:** The diagram can represent the interactions between the app and the notification system. It would show how the app sends notifications to users when new messages or updates are received, and how users interact with those notifications.

- **Error Handling:** The Sequence Diagram can depict the interactions related to error handling and exception scenarios. It would show how errors or exceptions are detected, reported, and handled within the app, including displaying error messages to users.

- **Logout and Session Management:** The diagram can illustrate the sequence of interactions involved in user logout and session management. It would show the steps taken to securely end the user session and perform any necessary cleanup or session-related tasks.

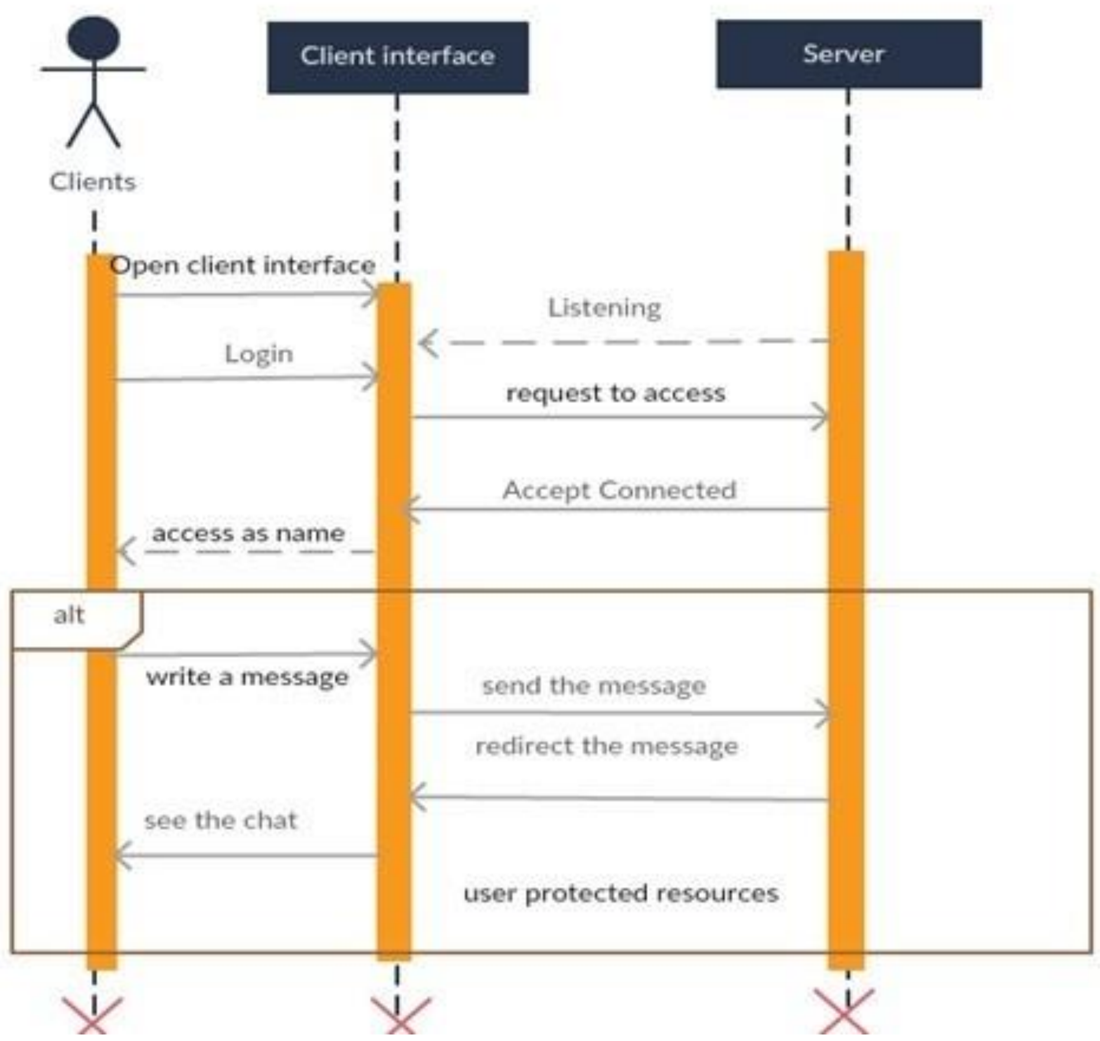


Figure 5 : Sequence Diagram

3.6 COLLABORATION DIAGRAM

A Collaboration Diagram is useful in the development of a chatting app for several reasons:

- **Visualizing Interactions:** A Collaboration Diagram provides a visual representation of how different objects or entities in the system collaborate and interact with each other. It helps developers understand the flow of information and the sequence of interactions between components, making it easier to identify potential issues or bottlenecks.

- **Clarifying Object Relationships**: The diagram illustrates the relationships and dependencies between objects or entities in the app. It helps in understanding how objects communicate and cooperate to accomplish specific tasks. This clarity assists in designing and implementing the app's architecture and ensures that the interactions between components are properly defined.

- **Identifying Communication Channels:** The Collaboration Diagram helps in identifying the communication channels between different objects or entities. It shows how messages or data are passed between components, clarifying the interfaces and API calls required for successful communication. This information is vital for developers when implementing the app's functionality.

- **Analyzing System Behavior:** By visualizing the interactions and collaborations between objects, the diagram enables developers to analyze the behavior of the system. It helps in identifying potential issues such as concurrency problems, race conditions, or communication conflicts. This analysis can lead to improvements in the app's performance and reliability.

- **Communication and Collaboration:** The Collaboration Diagram serves as a communication tool among developers, designers, and stakeholders. It provides a shared understanding of how different components work together, facilitating effective discussions and decision-making during the development process. It helps in aligning the development team's understanding of the app's architecture and functionality.

- **Test Case Generation:** The Collaboration Diagram can aid in generating test cases for the app. By examining the interactions between components, developers can identify the critical paths and scenarios that need to be tested. It helps in designing comprehensive test cases that cover all the necessary interactions and ensure the app functions as intended.

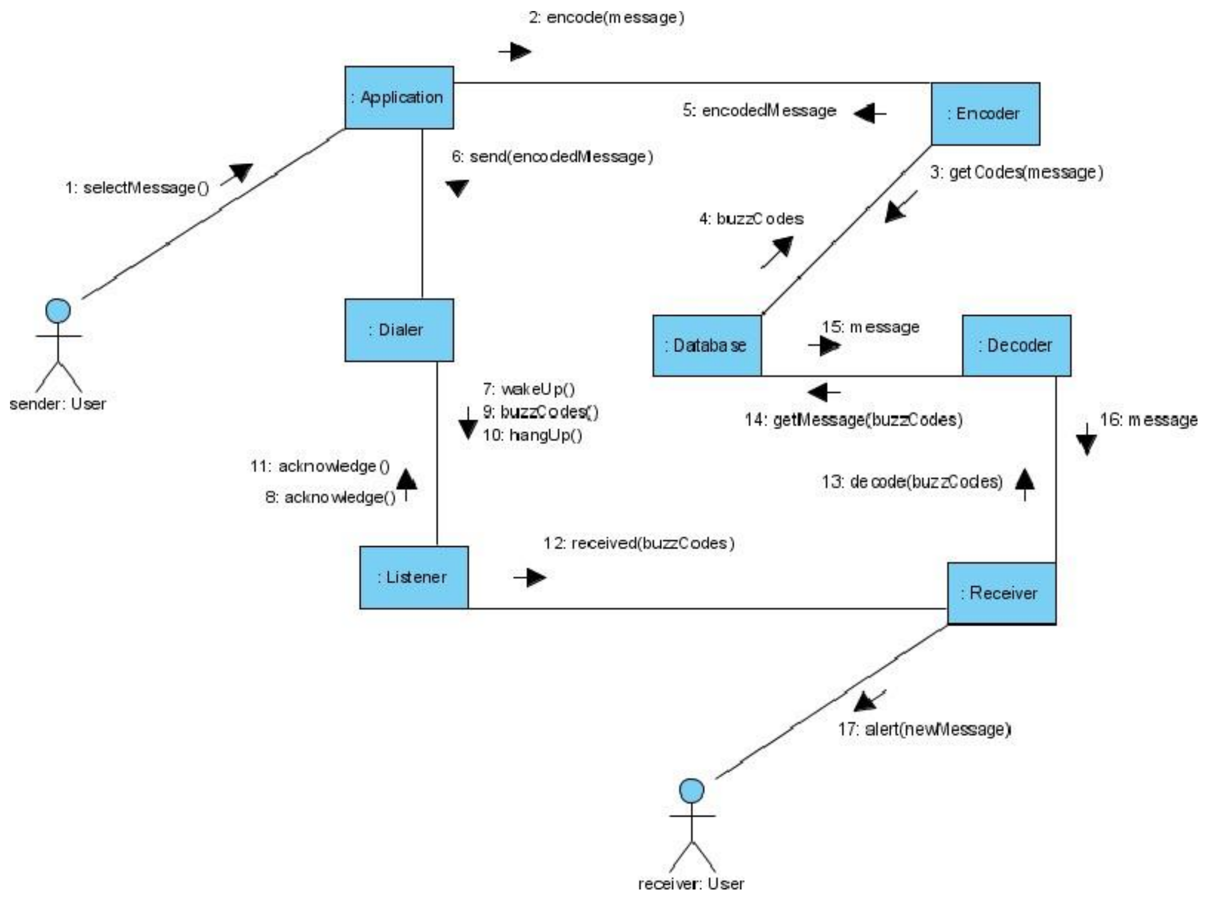


Figure 6 : Collaboration Diagram

3.7 COMPONENT DIAGRAM

A Component Diagram is useful in the development of a chatting app to depict the high-level structure and organization of the system, highlighting the various components and their interdependencies. Here's how a Component Diagram can be applied:

- **User Interface Component:** The Component Diagram can include the User Interface component, representing the front-end or client-side of the chatting app. It encompasses the graphical user interface (GUI), screens, forms, and user interaction elements.

- **Application Server Component:** The diagram can depict the Application Server component, representing the server-side logic of the chatting app. This component handles business logic, authentication, message routing, and other core functionalities.

- **Database Component:** The Component Diagram can include the Database component, representing the storage and retrieval of data in the app. It illustrates the database management system (DBMS) or data storage technology used, and the tables or collections storing user profiles, chats, messages, and other relevant data.

- **Messaging Service Component:** If the app includes real-time messaging capabilities, the diagram can show the Messaging Service component. This component manages the communication and exchange of messages between users, ensuring reliable and efficient message delivery.

- **Media Storage Component:** If multimedia sharing is supported, the Component Diagram can include the Media Storage component. It represents the storage infrastructure or cloud service used to store and retrieve media files, such as images or videos.
- **External Services or APIs:** The diagram can illustrate external services or APIs used by the app, such as authentication services (OAuth, OpenID), notification services (Push Notifications), or media processing services (image compression, video transcoding). These components depict the integration points with external systems.
- **Dependency Relationships:** The Component Diagram highlights the dependencies between components, indicating the required interfaces, dependencies, or communication channels between them. For example, the User Interface component may depend on the Application Server component for user authentication or message retrieval.
- **Deployment Environment:** The Component Diagram can also depict the deployment environment, illustrating the physical or virtual infrastructure where the components are deployed. This may include servers, cloud platforms, or other hosting environments.

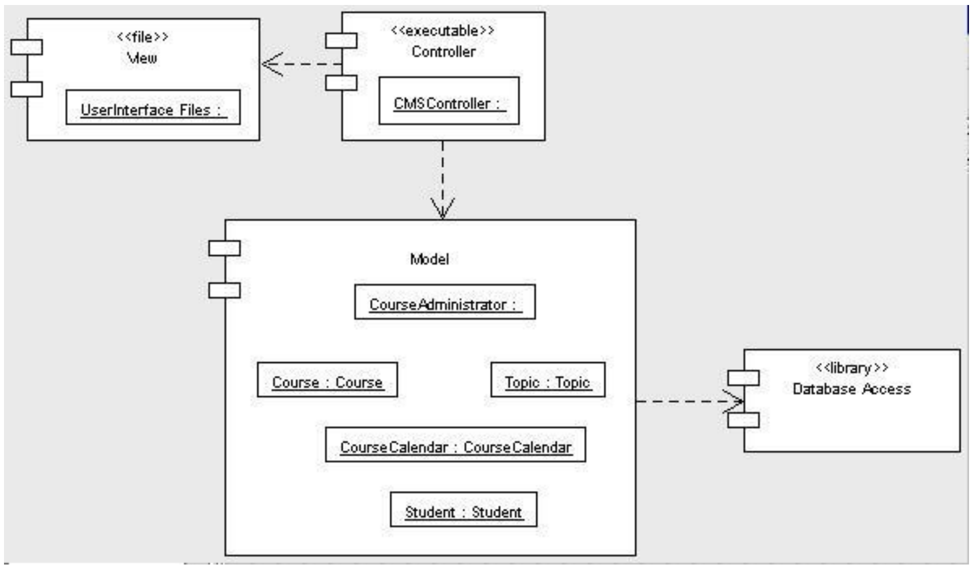
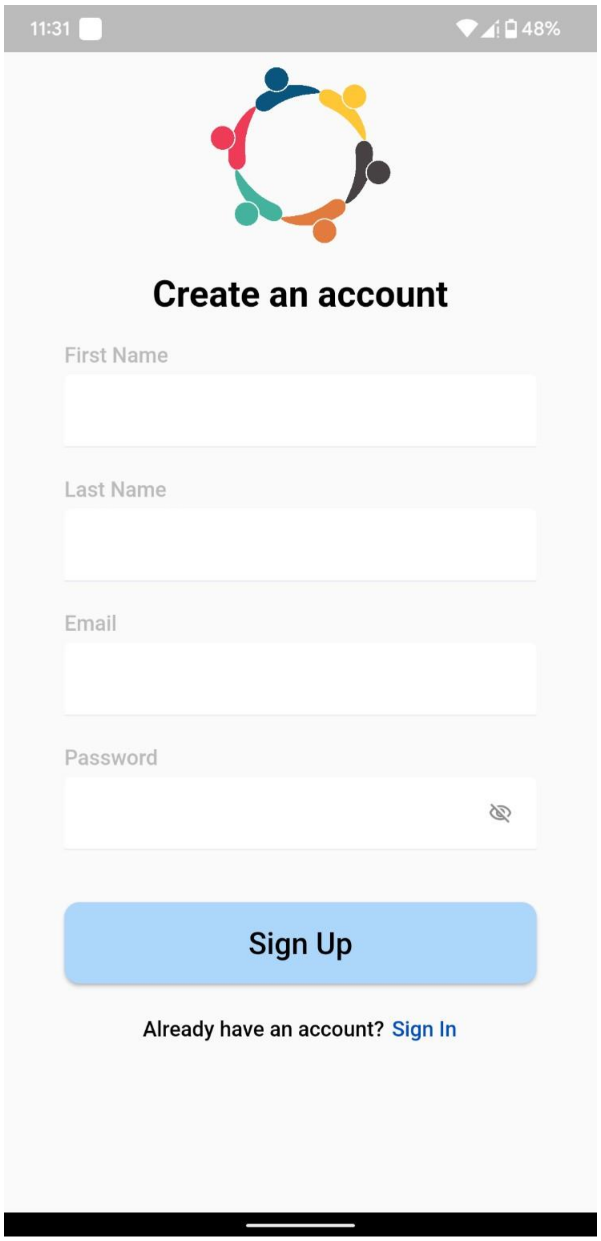


Figure 7 : Component Diagram

CHAPTER 4

FORM DESIGN


11:31 48%

Create an account

First Name

Last Name

Email

Password

Sign Up

Already have an account? Sign In

Figure 8 : Sign Up Page

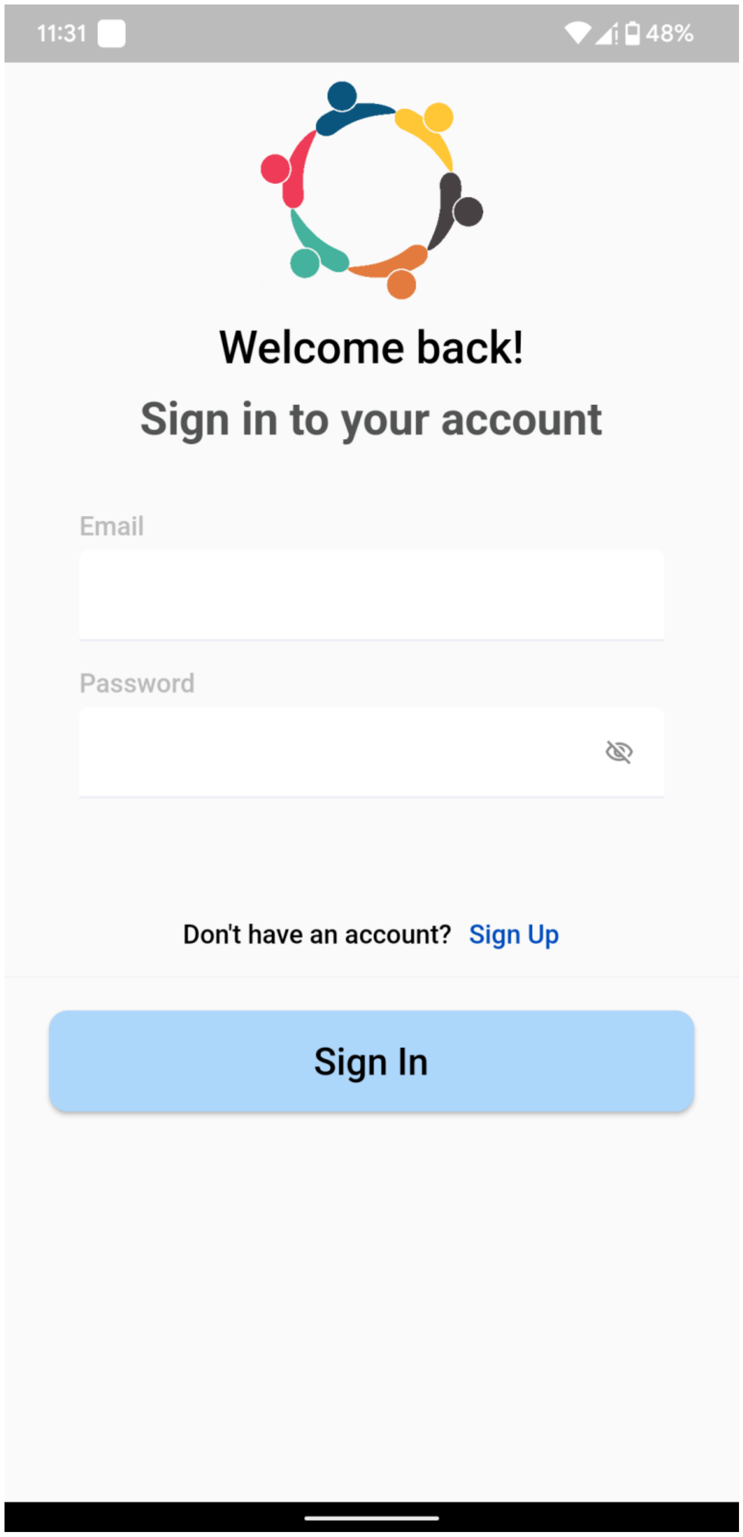
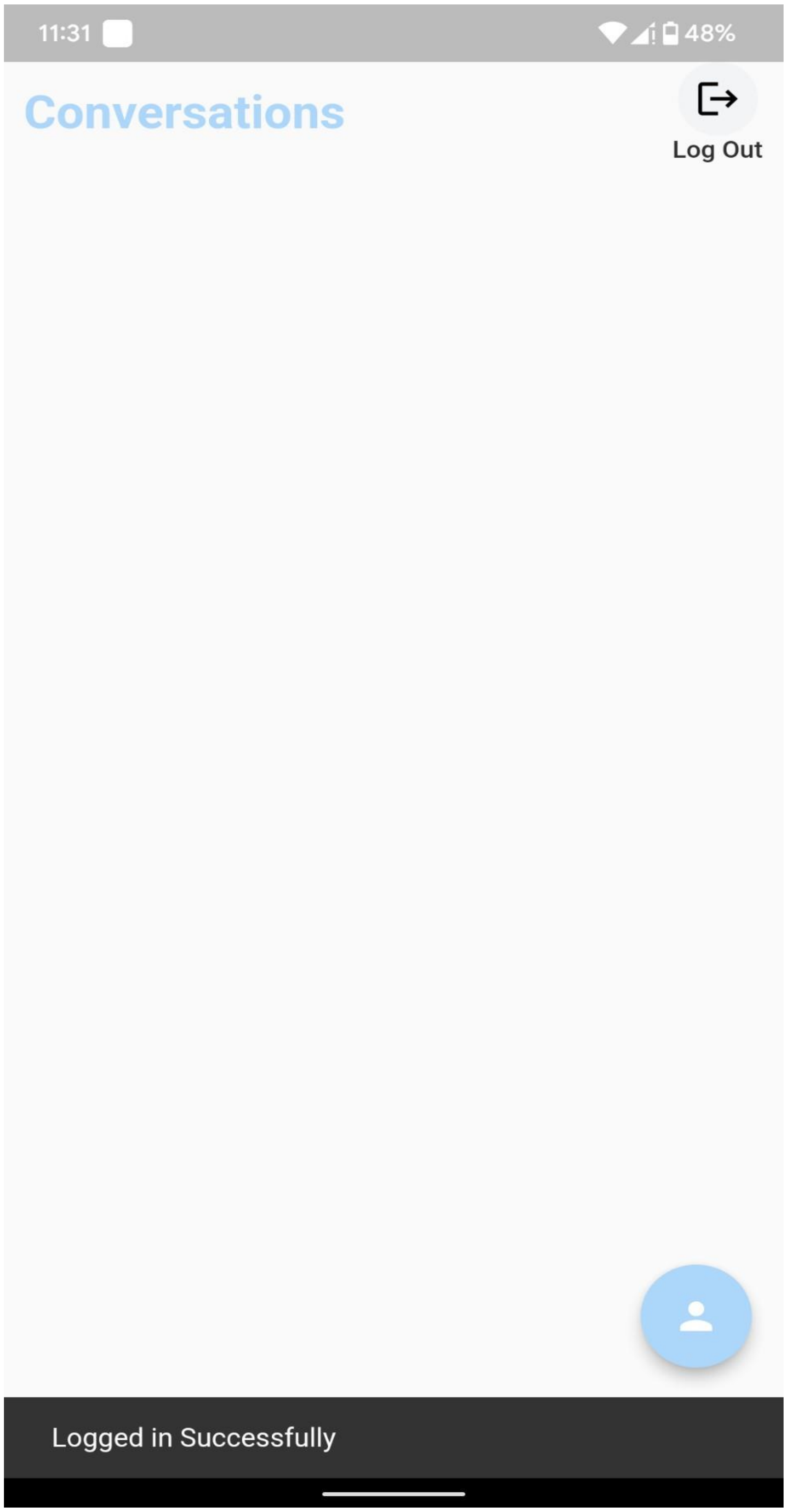


Figure 9 : Sign In Page



Logged in Successfully

Figure 10 : Logged In Page

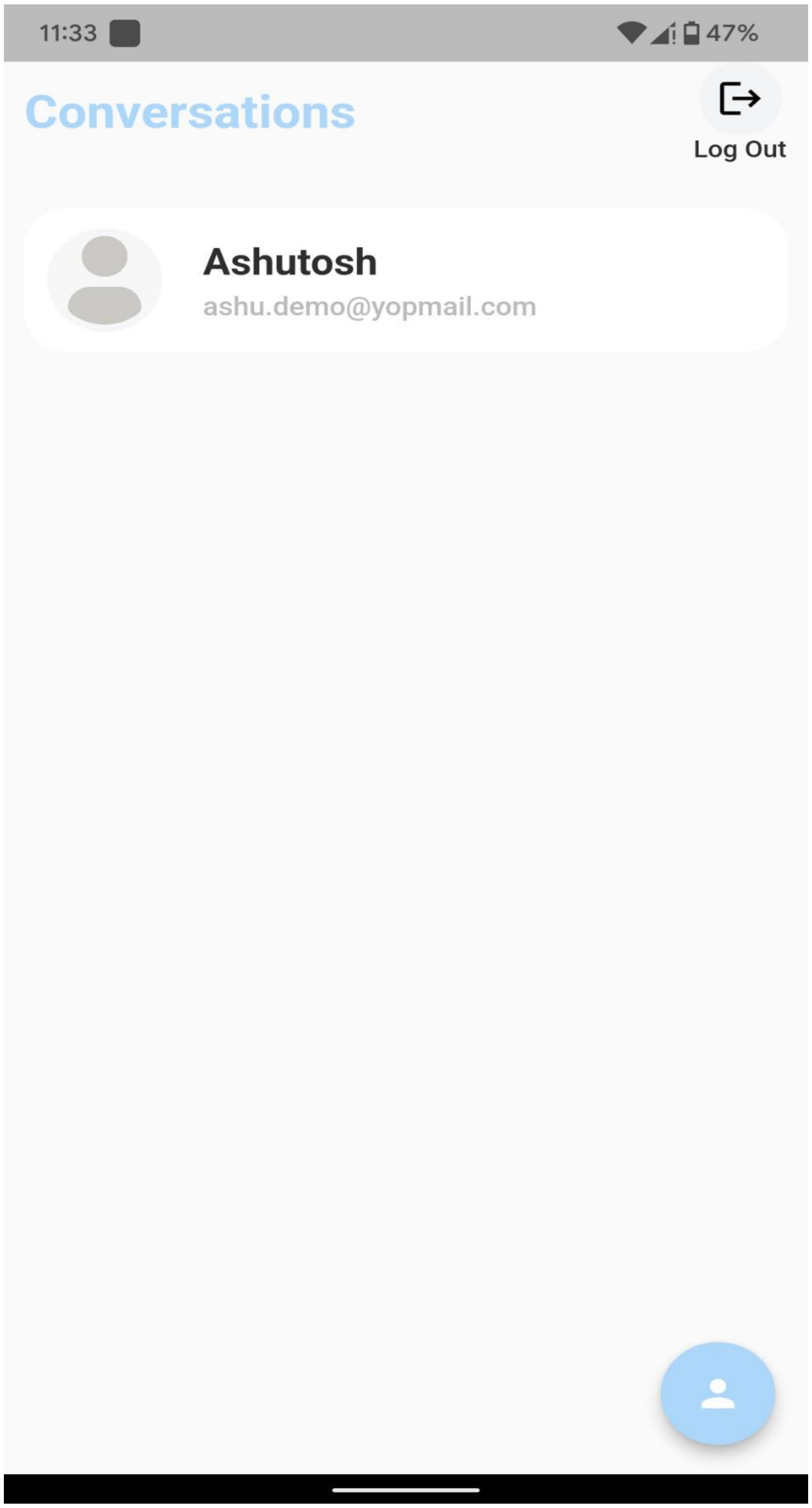
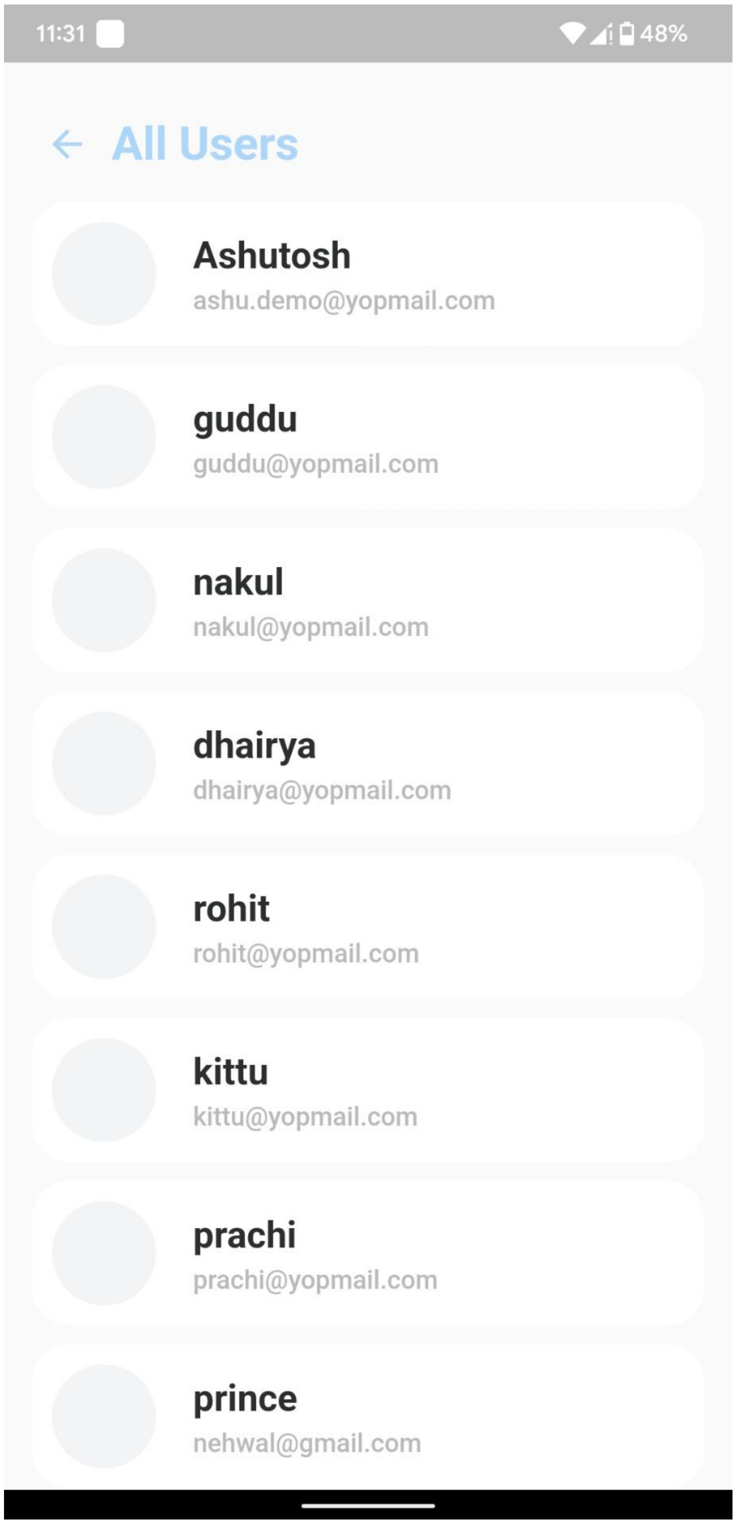


Figure 11 : Conversations Page


11:31 48%

All Users

Ashutosh
ashu.demo@yopmail.com

guddu
guddu@yopmail.com

nakul
nakul@yopmail.com

dhairya
dhairya@yopmail.com

rohit
rohit@yopmail.com

kittu
kittu@yopmail.com

prachi
prachi@yopmail.com

prince
nehwal@gmail.com

Figure 12 : All Users Page

11:31
47%
Ashutosh
type message

Figure 13 : Name Page

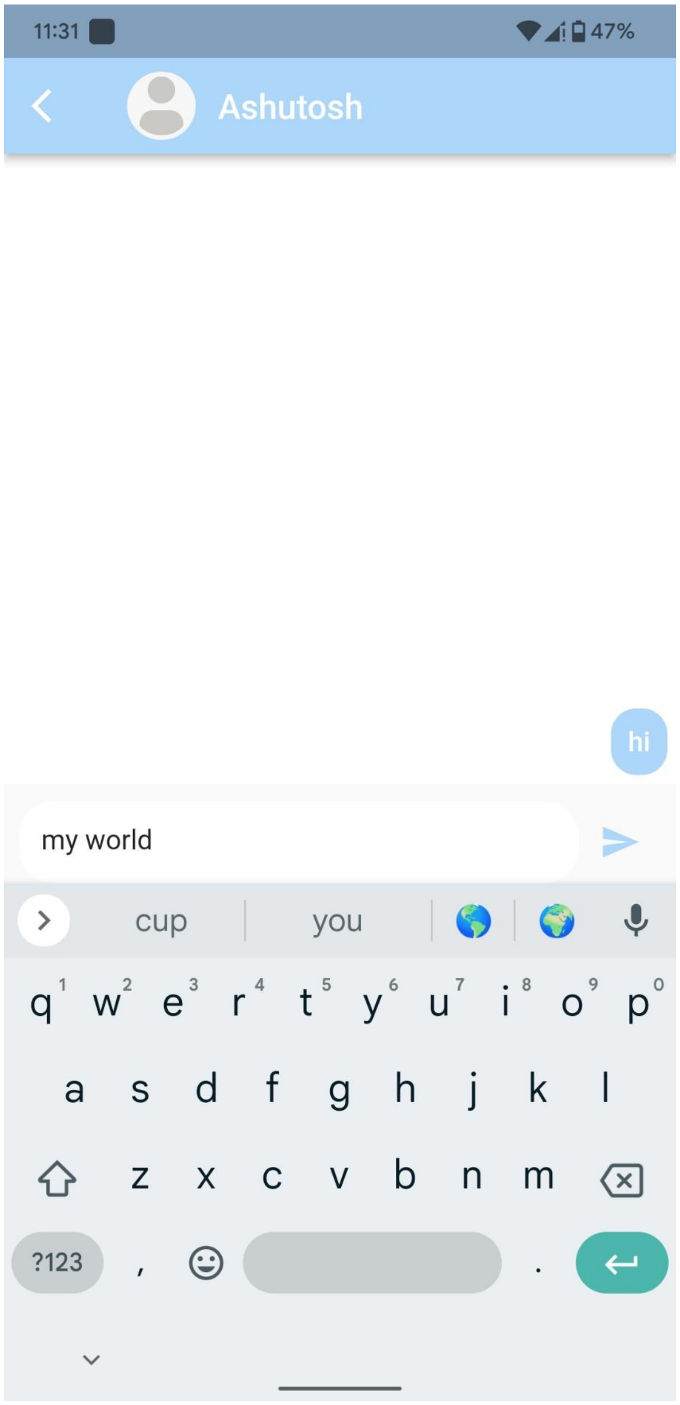


Figure 14 : Message Page

CHAPTER 5

TESTING

5.1 Functional Testing

Functional testing is highly valuable in the development of a chatting app as it ensures that the app's features and functionalities work correctly and meet the intended requirements. Here's how functional testing can be useful, along with examples of test cases:

1. **User Registration:**

   - **Test Case 1:** Verify that a new user can successfully register by entering valid registration details, such as a unique username, email address, and password.

   - **Test Case 2:** Validate that the app displays an error message when attempting to register with invalid or duplicate information.

   - **Test Case 3:** Ensure that the user's registration information is stored correctly in the database and can be retrieved for future logins.

2. **User Authentication and Login:**

   - **Test Case 1:** Confirm that users can log in with valid credentials and are granted access to the app's features and functionalities.

- **Test Case 2:** Verify that the app displays appropriate error messages when users enter incorrect login credentials or forget their password.
- **Test Case 3:** Test the "Remember Me" functionality, ensuring that users can stay logged in across app sessions when the option is selected.

3. **One-to-One Chat:**

- **Test Case 1:** Validate that users can start a one-to-one chat by selecting a contact and sending messages back and forth.
- **Test Case 2:** Ensure that messages are delivered and displayed accurately and in the correct order to both the sender and receiver.
- **Test Case 3:** Test the app's ability to handle multimedia content within one-to-one chats, such as sending and receiving photos or videos.
- **Selection of Operating System:** Our website is platform-independent, so it does not depend on the operating system.

4. **Group Chat:**

- **Test Case 1:** Verify that users can create a group chat and invite other users to join the group.
- **Test Case 2:** Validate that messages sent in a group chat are received by all members of the group and displayed correctly.

- **Test Case 3:** Test the functionality of adding or removing participants from a group chat, ensuring that updates are reflected accurately.

5. **Multimedia Sharing:**

- **Test Case 1:** Ensure that users can upload and share multimedia content, such as photos or videos, within chats or groups.

- **Test Case 2:** Validate that shared media files are displayed correctly to the recipients and can be downloaded or viewed as intended.

- **Test Case 3:** Test the app's ability to handle different file formats and sizes when sharing multimedia content.

6. **Notifications:**

- **Test Case 1:** Validate that users receive push notifications for new messages or updates, even when the app is in the background or the device is asleep.

-

- **Test Case 2:** Verify that notifications are delivered accurately and in a timely manner without any delay or loss.

- **Test Case 3:** Test the behavior of notifications when the user is actively using the app, ensuring they don't disrupt the user experience.

5.2 NON-FUNCTIONAL TESTING

Non-functional testing is essential in the development of a chatting app as it focuses on evaluating aspects of the app beyond its specific features. It assesses the app's performance, usability, security, and other non-functional aspects. Here's how non-functional testing can be useful, along with examples of test cases:

1. **Performance Testing:**

- **Test Case 1:** Evaluate the app's response time for common operations such as sending messages, loading chats, or retrieving user profiles.

- **Test Case 2:** Measure the app's scalability by simulating a high number of concurrent users and monitoring its performance under load.

- **Test Case 3:** Verify the app's ability to handle a large volume of messages, ensuring that it doesn't degrade in performance.

2. **Usability Testing:**

- **Test Case 1:** Assess the app's ease of use and intuitiveness by observing users' interactions with the user interface and gathering feedback.

- **Test Case 2:** Validate that the app adheres to user interface design guidelines, including consistent layout, clear labeling, and intuitive navigation.

- **Test Case 3:** Test the app's accessibility features, ensuring compatibility with screen readers, font size adjustments, and color contrast options.

3. **Security Testing:**

- **Test Case 1:** Assess the app's resistance to common security vulnerabilities, such as SQL injection, cross-site scripting (XSS), or session hijacking.

- **Test Case 2:** Validate the effectiveness of encryption mechanisms for sensitive data transmission, such as user credentials or media files.

- **Test Case 3:** Verify that user authentication and authorization mechanisms prevent unauthorized access to chats or user information.

4. **Compatibility Testing:**

- **Test Case 1:** Validate the app's compatibility with different operating systems, such as iOS and Android, and different versions of each platform.

- **Test Case 2:** Test the app's compatibility with various web browsers, ensuring consistent functionality and appearance across different browsers and versions.

- **Test Case 3:** Verify the app's compatibility with different screen resolutions and device sizes to ensure a consistent user experience.

5. **Reliability Testing:**

- **Test Case 1:** Verify the app's stability by conducting prolonged testing sessions and monitoring for crashes or unexpected behavior.

- **Test Case 2:** Validate the app's ability to recover from unexpected events, such as network interruptions or server failures, without data loss or corruption.

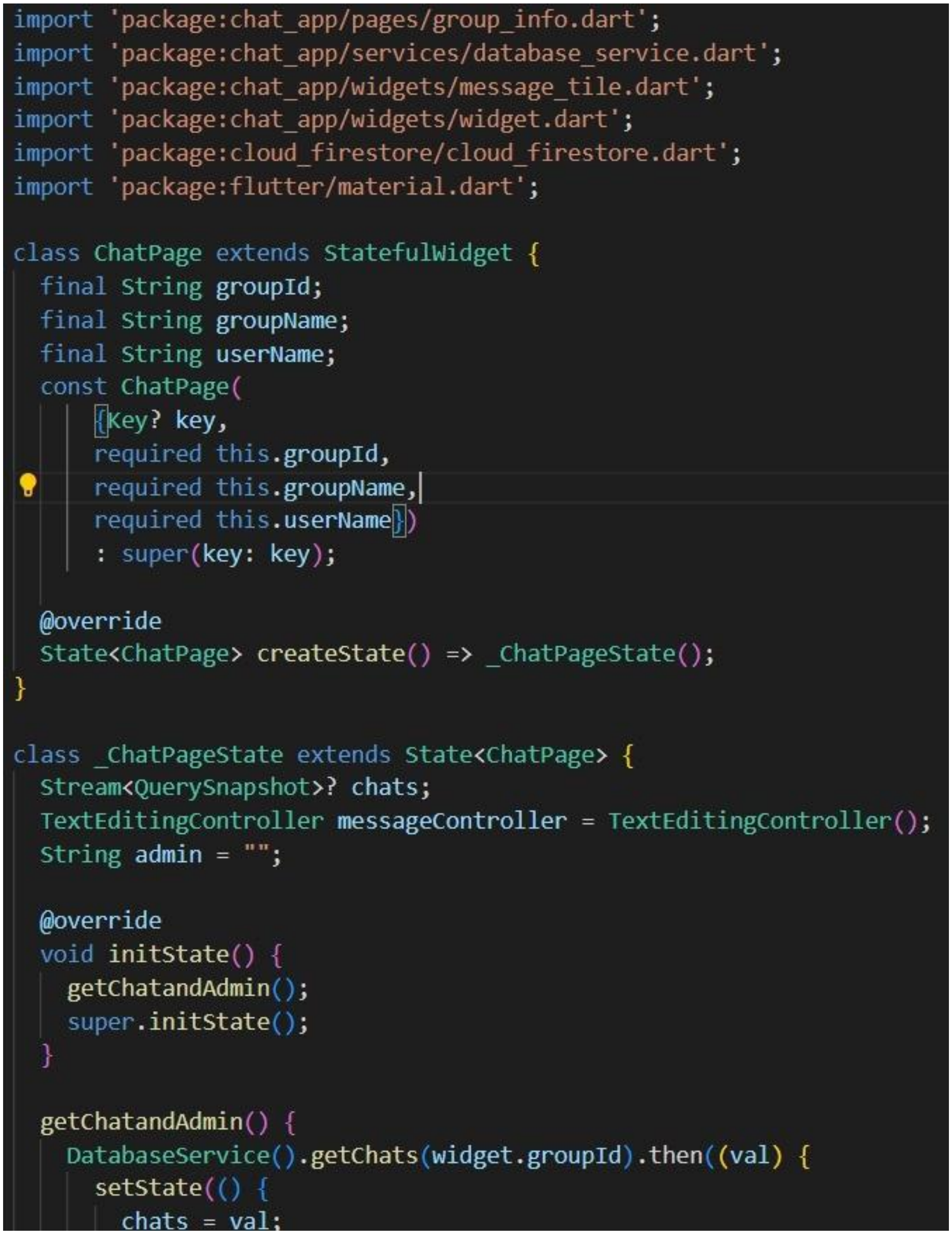
- **Test Case 3:** Test the app's behavior under low or unstable network conditions, ensuring it handles connectivity issues gracefully.

6. **Load Testing:**

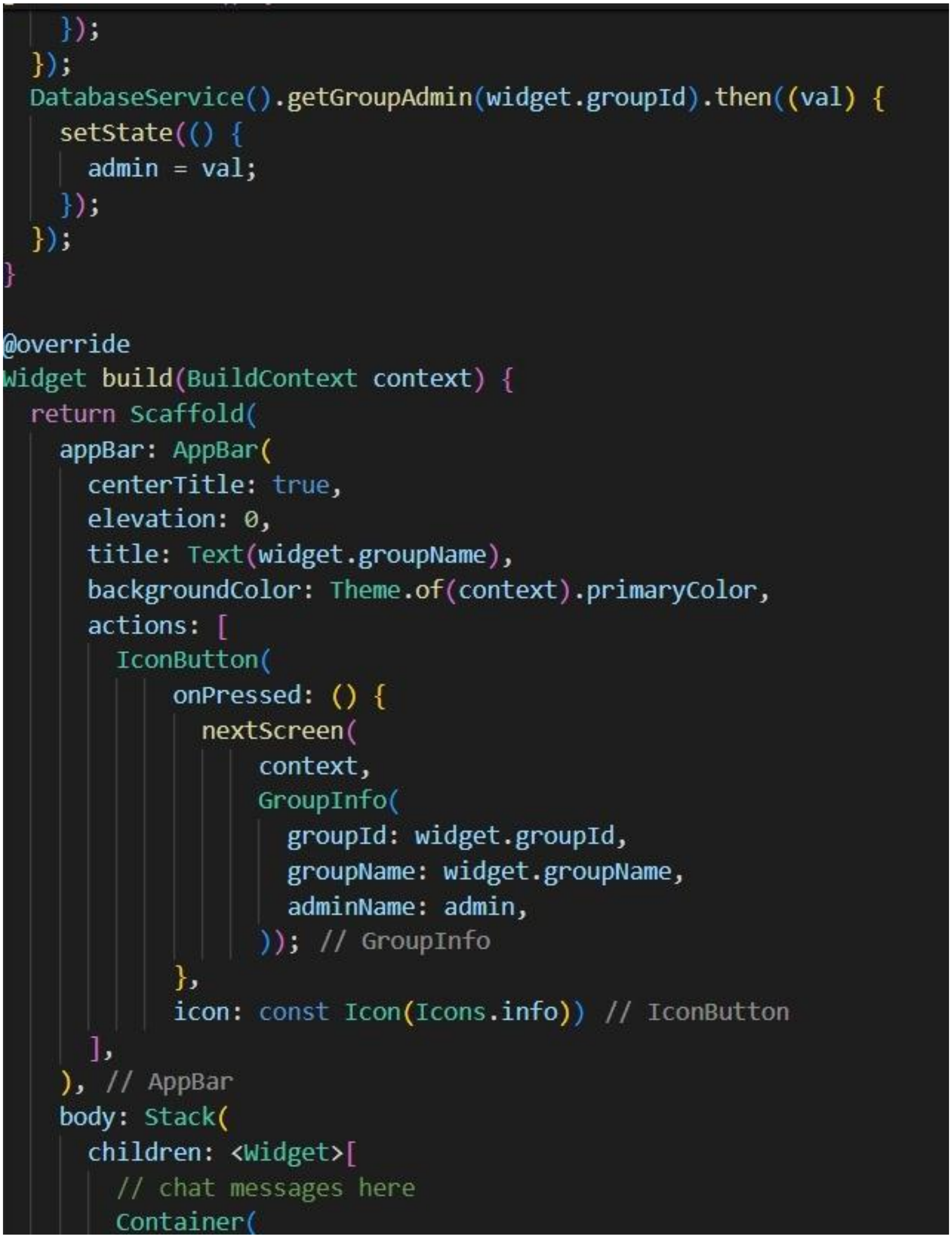
- **Test Case 1:** Assess the app's performance and stability under heavy loads by simulating a large number of users and monitoring response times and resource usage.

- **Test Case 2:** Validate that the app's infrastructure, including servers and databases, can handle the expected load without performance degradation.

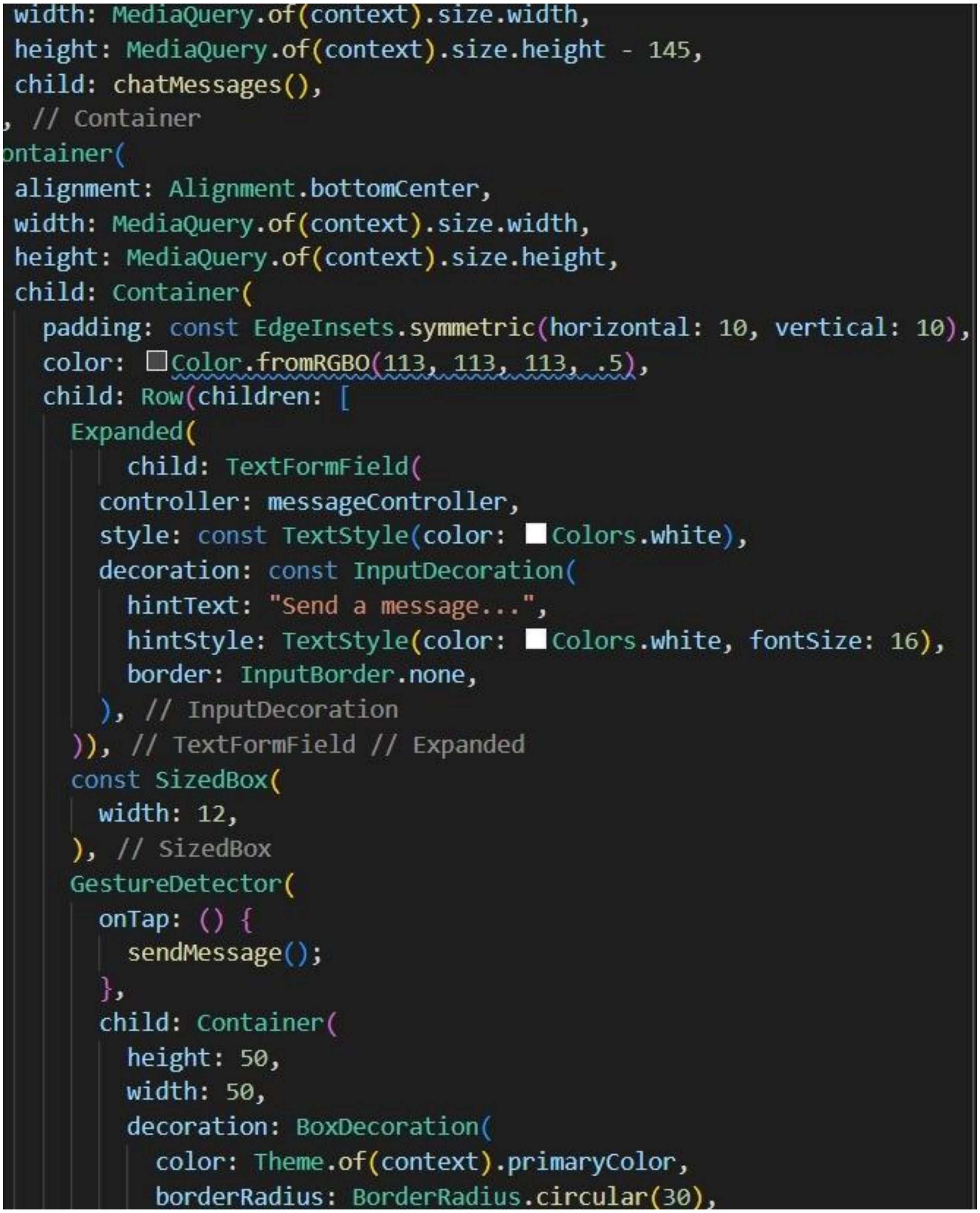
- **Test Case 3:** Test the app's behaviour when multiple users perform simultaneous actions, such as sending messages or sharing media.



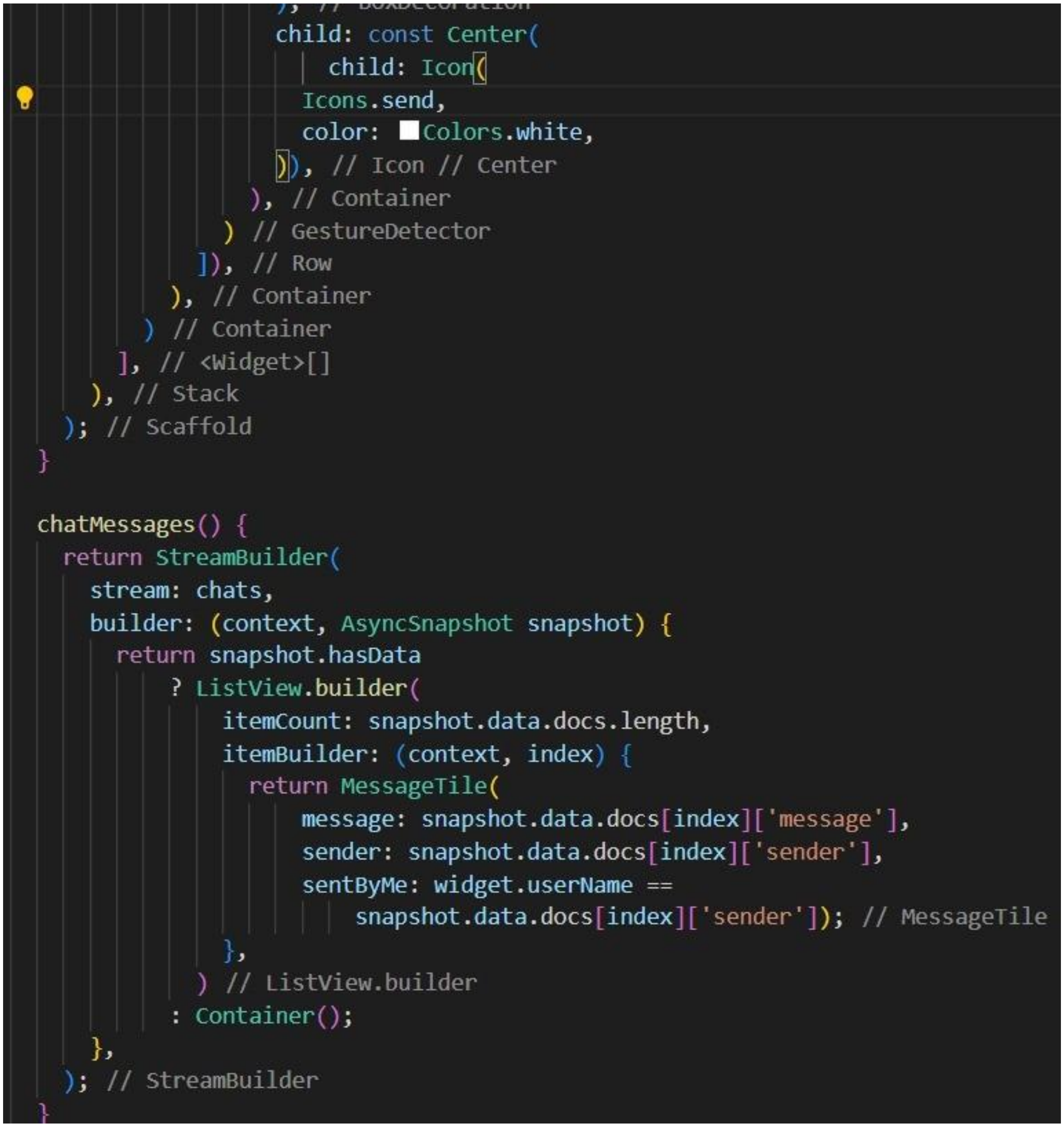
```
import 'package:chat_app/pages/group_info.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:chat_app/widgets/message_tile.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class ChatPage extends StatefulWidget {
  final String groupId;
  final String groupName;
  final String userName;
  const ChatPage(
      {Key? key,
      required this.groupId,
      required this.groupName,
      required this.userName})
      : super(key: key);

  @override
  State<ChatPage> createState() => _ChatPageState();
}

class _ChatPageState extends State<ChatPage> {
  Stream<QuerySnapshot>? chats;
  TextEditingController messageController = TextEditingController();
  String admin = "";

  @override
  void initState() {
    getChatandAdmin();
    super.initState();
  }

  getChatandAdmin() {
    DatabaseService().getChats(widget.groupId).then((val) {
      setState(() {
        chats = val;
```



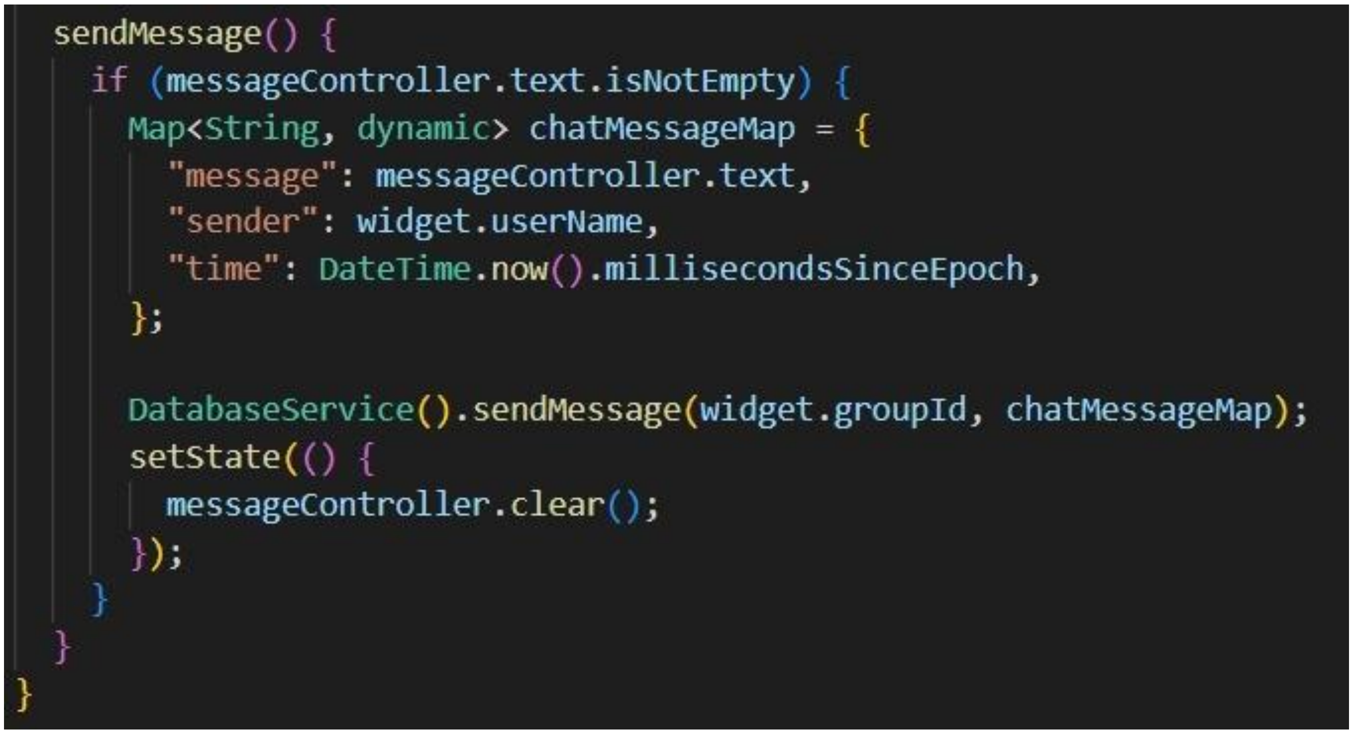
```
    });
  });
  DatabaseService().getGroupAdmin(widget.groupId).then((val) {
    setState(() {
      admin = val;
    });
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      centerTitle: true,
      elevation: 0,
      title: Text(widget.groupName),
      backgroundColor: Theme.of(context).primaryColor,
      actions: [
        IconButton(
            onPressed: () {
              nextScreen(
                  context,
                  GroupInfo(
                    groupId: widget.groupId,
                    groupName: widget.groupName,
                    adminName: admin,
                  )); // GroupInfo
            },
            icon: const Icon(Icons.info)) // IconButton
      ],
    ), // AppBar
    body: Stack(
      children: <Widget>[
        // chat messages here
        Container(
```

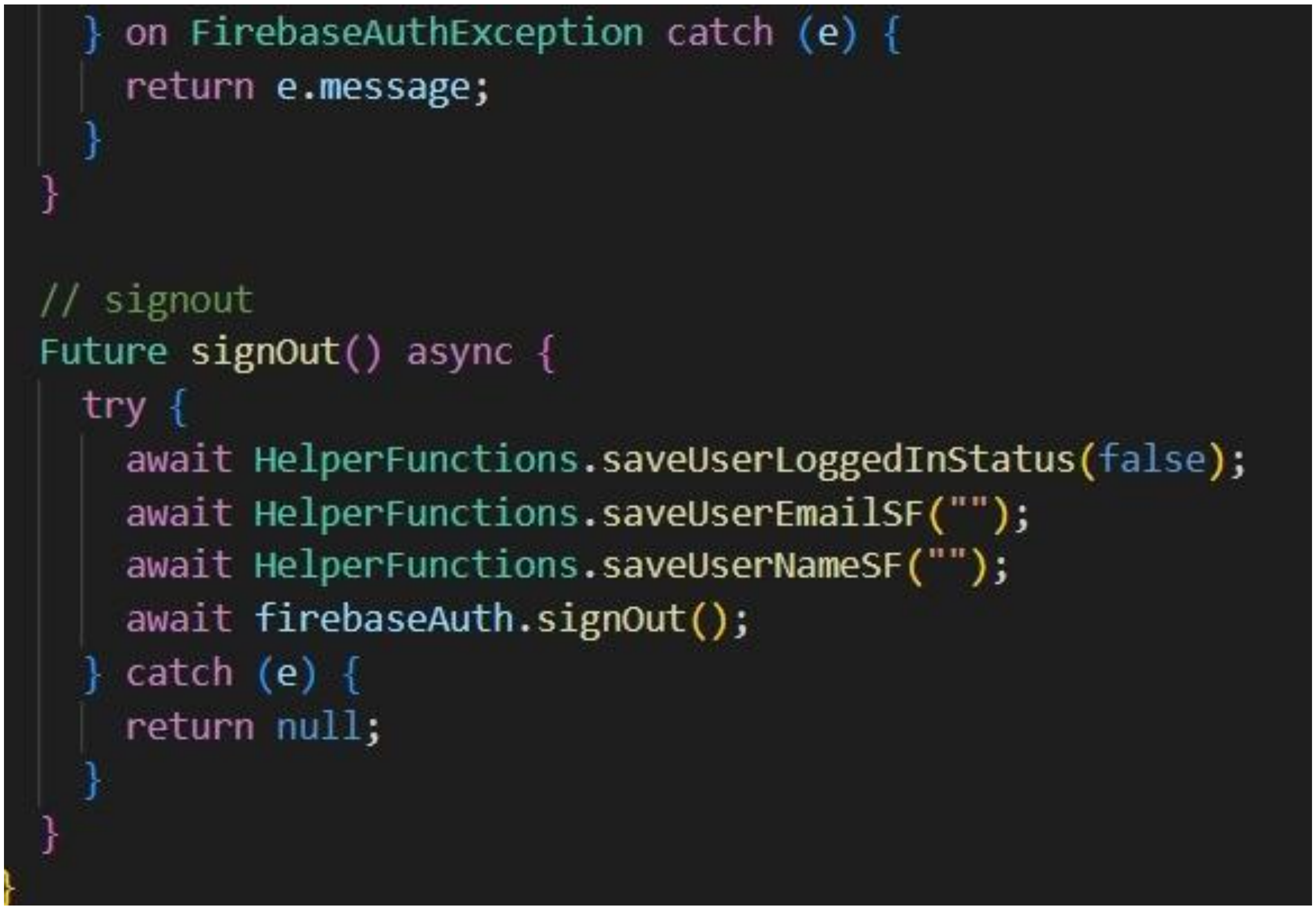


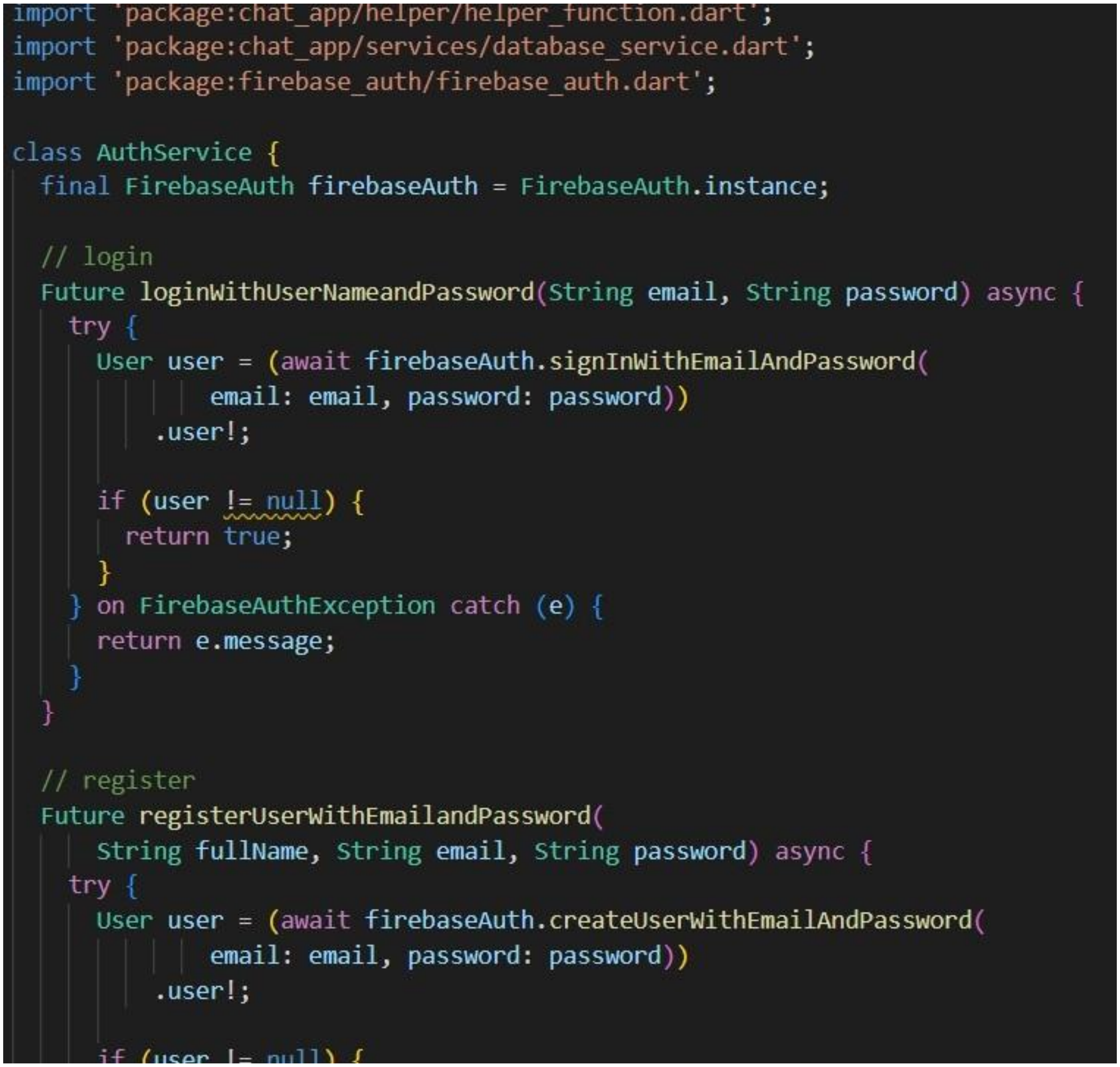
```
 width: MediaQuery.of(context).size.width,
 height: MediaQuery.of(context).size.height - 145,
 child: chatMessages(),
, // Container
ontainer(
 alignment: Alignment.bottomCenter,
 width: MediaQuery.of(context).size.width,
 height: MediaQuery.of(context).size.height,
 child: Container(
   padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 10),
   color: Color.fromRGBO(113, 113, 113, .5),
   child: Row(children: [
     Expanded(
         child: TextFormField(
       controller: messageController,
       style: const TextStyle(color: Colors.white),
       decoration: const InputDecoration(
         hintText: "Send a message...",
         hintStyle: TextStyle(color: Colors.white, fontSize: 16),
         border: InputBorder.none,
       ), // InputDecoration
     )), // TextFormField // Expanded
     const SizedBox(
       width: 12,
     ), // SizedBox
     GestureDetector(
       onTap: () {
         sendMessage();
       },
       child: Container(
         height: 50,
         width: 50,
         decoration: BoxDecoration(
           color: Theme.of(context).primaryColor,
           borderRadius: BorderRadius.circular(30),
```



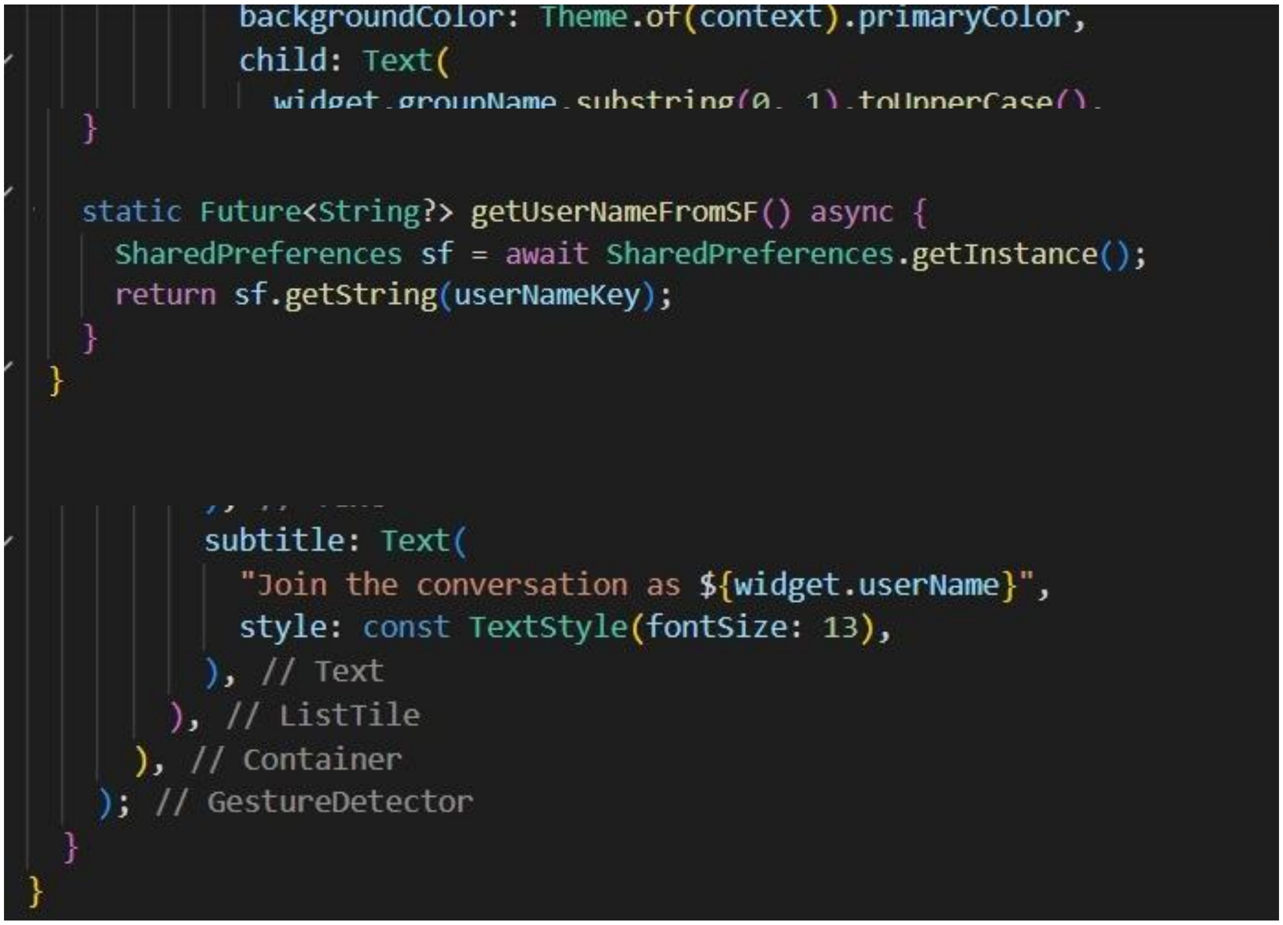
```
                    child: const Center(
                      child: Icon(
                      Icons.send,
                      color: Colors.white,
                    )), // Icon // Center
                  ), // Container
                ) // GestureDetector
              ]), // Row
            ), // Container
          ) // Container
        ], // <Widget>[]
      ), // Stack
    ); // Scaffold
  }

  chatMessages() {
    return StreamBuilder(
      stream: chats,
      builder: (context, AsyncSnapshot snapshot) {
        return snapshot.hasData
            ? ListView.builder(
                itemCount: snapshot.data.docs.length,
                itemBuilder: (context, index) {
                  return MessageTile(
                      message: snapshot.data.docs[index]['message'],
                      sender: snapshot.data.docs[index]['sender'],
                      sentByMe: widget.userName ==
                          snapshot.data.docs[index]['sender']); // MessageTile
                },
              ) // ListView.builder
            : Container();
      },
    ); // StreamBuilder
  }
```


```
  sendMessage() {
    if (messageController.text.isNotEmpty) {
      Map<String, dynamic> chatMessageMap = {
        "message": messageController.text,
        "sender": widget.userName,
        "time": DateTime.now().millisecondsSinceEpoch,
      };

      DatabaseService().sendMessage(widget.groupId, chatMessageMap);
      setState(() {
        messageController.clear();
      });
    }
  }
}
```

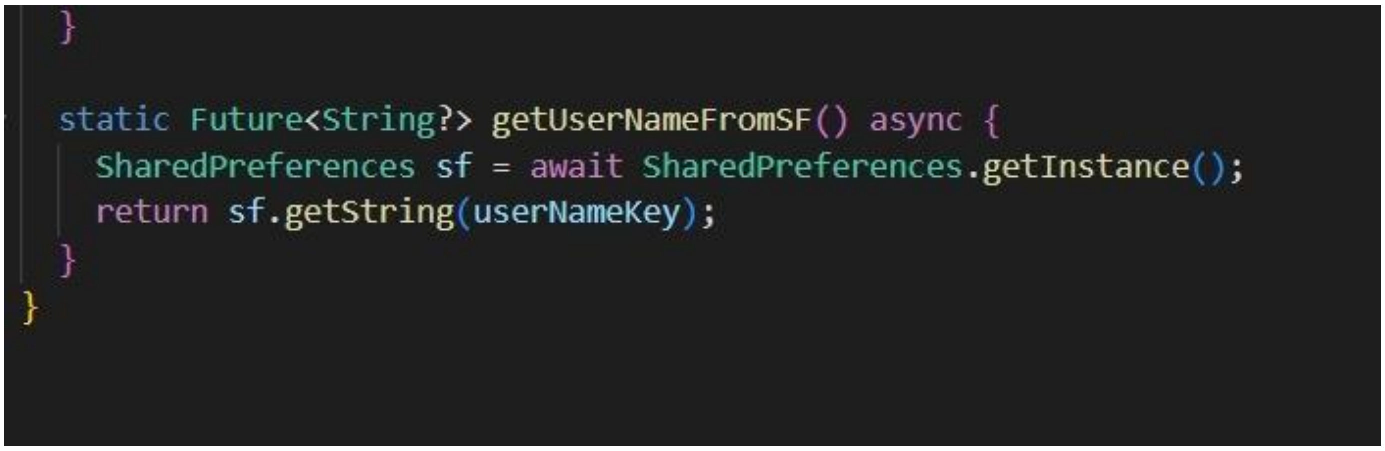

```
    } on FirebaseAuthException catch (e) {
      return e.message;
    }
  }

  // signout
  Future signOut() async {
    try {
      await HelperFunctions.saveUserLoggedInStatus(false);
      await HelperFunctions.saveUserEmailSF("");
      await HelperFunctions.saveUserNameSF("");
      await firebaseAuth.signOut();
    } catch (e) {
      return null;
    }
  }
}
```



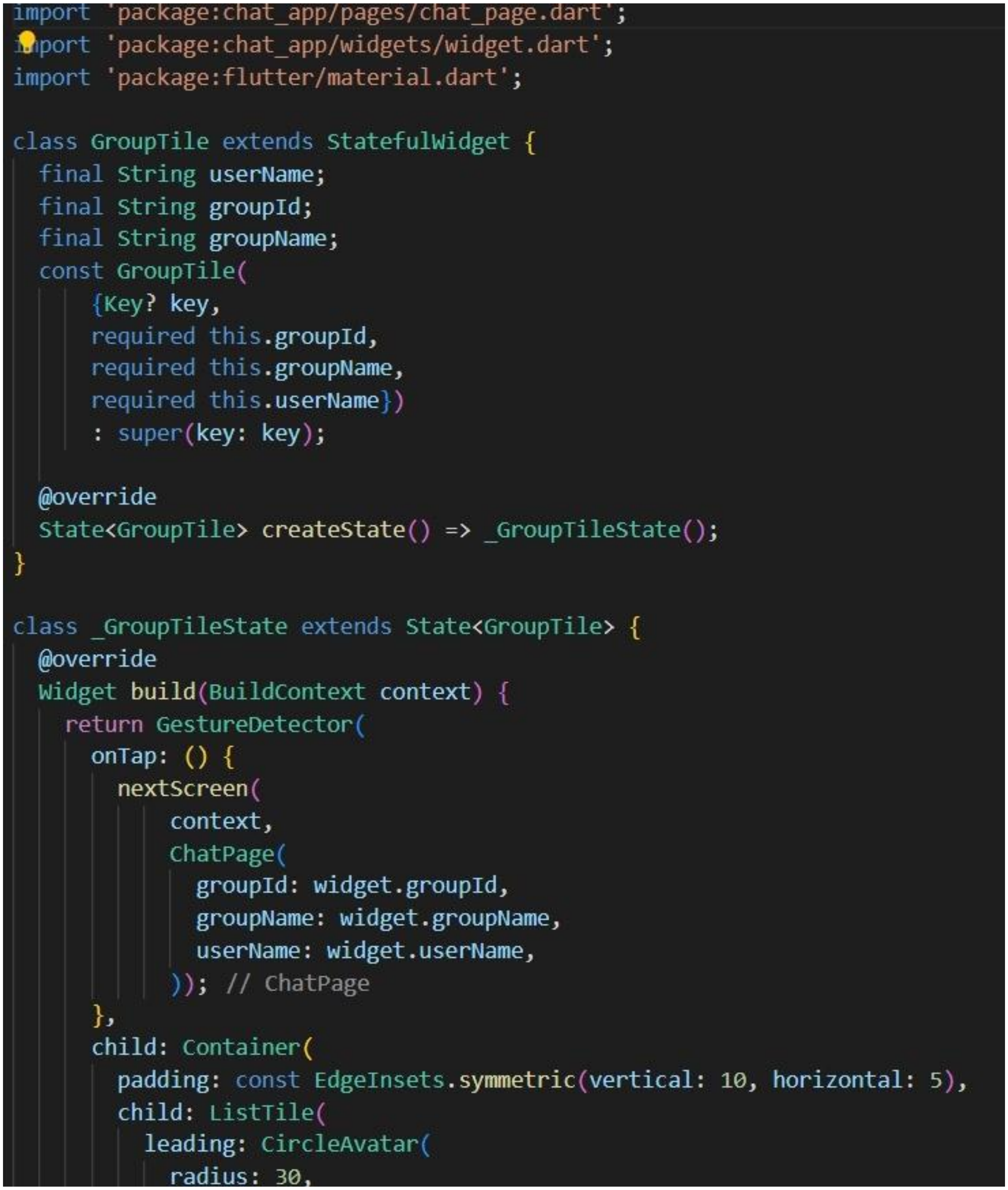
```
import 'package:chat_app/helper/helper_function.dart';
import 'package:chat_app/services/database_service.dart';
import 'package:firebase_auth/firebase_auth.dart';

class AuthService {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;

  // login
  Future loginWithUserNameandPassword(String email, String password) async {
    try {
      User user = (await firebaseAuth.signInWithEmailAndPassword(
                  email: email, password: password))
              .user!;

      if (user != null) {
        return true;
      }
    } on FirebaseAuthException catch (e) {
      return e.message;
    }
  }

  // register
  Future registerUserWithEmailandPassword(
      String fullName, String email, String password) async {
    try {
      User user = (await firebaseAuth.createUserWithEmailAndPassword(
                  email: email, password: password))
              .user!;

      if (user != null) {
```



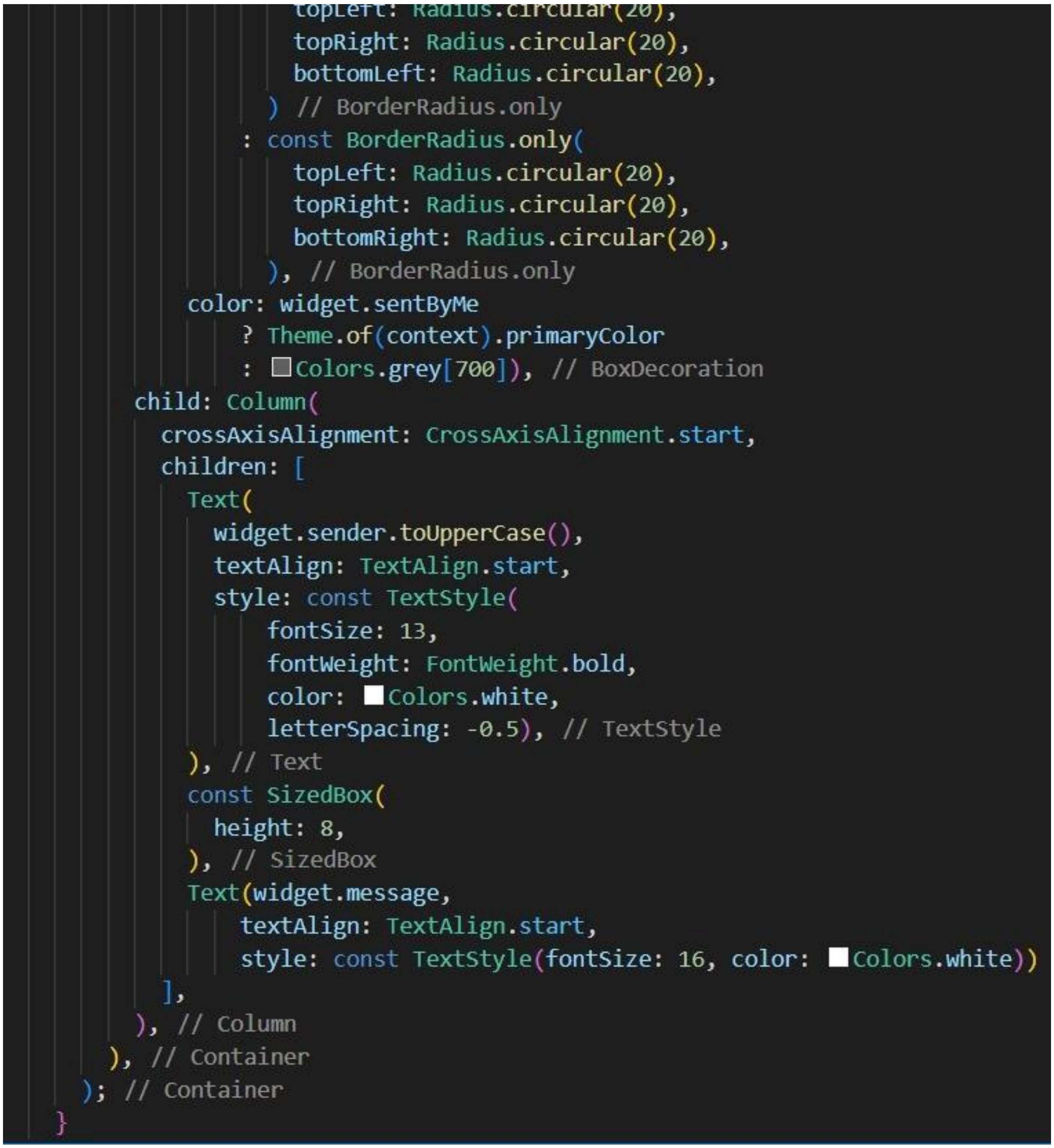
```
                backgroundColor: Theme.of(context).primaryColor,
                child: Text(
                  widget.groupName.substring(0, 1).toUpperCase(),
    }

    static Future<String?> getUserNameFromSF() async {
      SharedPreferences sf = await SharedPreferences.getInstance();
      return sf.getString(userNameKey);
    }
  }

              subtitle: Text(
                "Join the conversation as ${widget.userName}",
                style: const TextStyle(fontSize: 13),
              ), // Text
            ), // ListTile
          ), // Container
        ); // GestureDetector
      }
    }
```



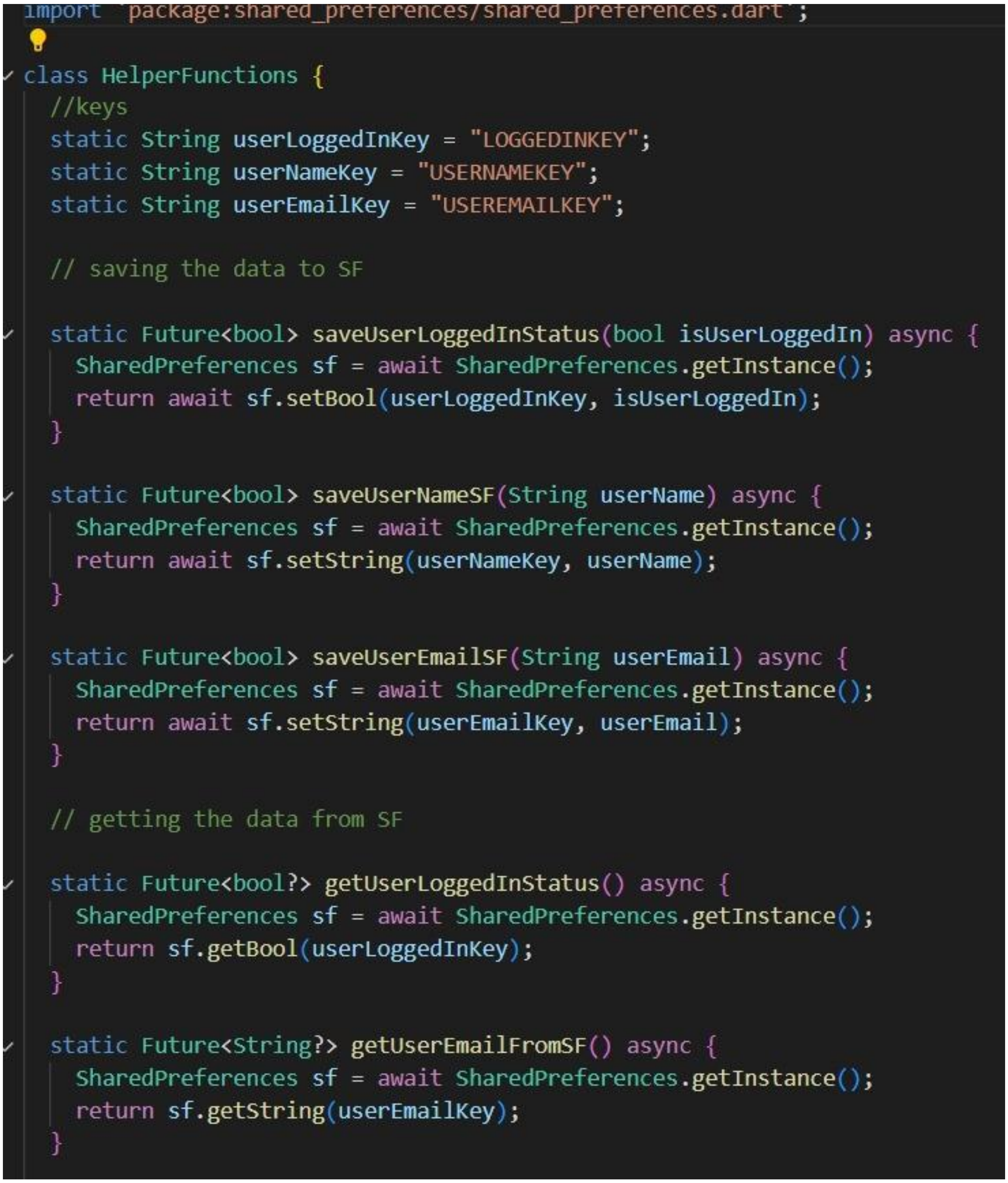
```
    }

    static Future<String?> getUserNameFromSF() async {
      SharedPreferences sf = await SharedPreferences.getInstance();
      return sf.getString(userNameKey);
    }
  }
```



```
import 'package:chat_app/pages/chat_page.dart';
import 'package:chat_app/widgets/widget.dart';
import 'package:flutter/material.dart';

class GroupTile extends StatefulWidget {
  final String userName;
  final String groupId;
  final String groupName;
  const GroupTile(
      {Key? key,
      required this.groupId,
      required this.groupName,
      required this.userName})
      : super(key: key);

  @override
  State<GroupTile> createState() => _GroupTileState();
}

class _GroupTileState extends State<GroupTile> {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        nextScreen(
            context,
            ChatPage(
              groupId: widget.groupId,
              groupName: widget.groupName,
              userName: widget.userName,
            )); // ChatPage
      },
      child: Container(
        padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 5),
        child: ListTile(
          leading: CircleAvatar(
            radius: 30,
```



```
                  topLeft: Radius.circular(20),
                  topRight: Radius.circular(20),
                  bottomLeft: Radius.circular(20),
                ) // BorderRadius.only
              : const BorderRadius.only(
                  topLeft: Radius.circular(20),
                  topRight: Radius.circular(20),
                  bottomRight: Radius.circular(20),
                ), // BorderRadius.only
          color: widget.sentByMe
              ? Theme.of(context).primaryColor
              : Colors.grey[700]), // BoxDecoration
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            widget.sender.toUpperCase(),
            textAlign: TextAlign.start,
            style: const TextStyle(
                fontSize: 13,
                fontWeight: FontWeight.bold,
                color: Colors.white,
                letterSpacing: -0.5), // TextStyle
          ), // Text
          const SizedBox(
            height: 8,
          ), // SizedBox
          Text(widget.message,
              textAlign: TextAlign.start,
              style: const TextStyle(fontSize: 16, color: Colors.white))
        ],
      ), // Column
    ), // Container
  ); // Container
}
```



```dart
import 'package:shared_preferences/shared_preferences.dart';

class HelperFunctions {
  //keys
  static String userLoggedInKey = "LOGGEDINKEY";
  static String userNameKey = "USERNAMEKEY";
  static String userEmailKey = "USEREMAILKEY";

  // saving the data to SF

  static Future<bool> saveUserLoggedInStatus(bool isUserLoggedIn) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setBool(userLoggedInKey, isUserLoggedIn);
  }

  static Future<bool> saveUserNameSF(String userName) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userNameKey, userName);
  }

  static Future<bool> saveUserEmailSF(String userEmail) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userEmailKey, userEmail);
  }

  // getting the data from SF

  static Future<bool?> getUserLoggedInStatus() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getBool(userLoggedInKey);
  }

  static Future<String?> getUserEmailFromSF() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getString(userEmailKey);
  }
```

CHAPTER 6

FUTURE SCOPE

The future scope of a friendly chat application is quite promising, with several potential avenues for development and innovation:

Enhanced User Experience: Continuously improving user experience will be a key focus, with features like intuitive interfaces, personalized recommendations, and seamless integration with other platforms.

AI-Powered Assistance: Integrating AI-powered chatbots can enhance user engagement by providing real-time assistance, answering queries, and even offering personalized recommendations based on user preferences and behavior.

Multimedia Integration: Future chat applications will likely support a wide range of multimedia content, including high-definition images, videos, GIFs, stickers, and augmented reality (AR) elements, enriching the communication experience.

Cross-Platform Compatibility: Ensuring compatibility across various devices and platforms, including smartphones, tablets, desktops, and wearables, will be crucial to cater to diverse user preferences and maximize accessibility.

Privacy and Security: With growing concerns about data privacy and security, future chat applications will need to implement robust encryption techniques, privacy controls, and secure authentication mechanisms to safeguard user data and ensure confidentiality.

Social Integration: Integrating social media features such as user profiles, status updates, and group chats can enhance connectivity and foster a sense of community among users.

Localization and Globalization: Adapting the chat application to different languages, cultures, and regional preferences will be essential for expanding its user base globally and catering to diverse audiences.

Voice and Video Communication: Integrating high-quality voice and video calling features within the chat application can offer users more ways to communicate effectively, bridging the gap between text-based and real-time communication.

Gamification and Rewards: Incorporating gamification elements such as achievements, badges, and rewards can increase user engagement and retention, making the chat application more enjoyable and addictive.

Monetization Strategies: Exploring various monetization strategies such as in-app purchases, premium features, subscription models, and targeted advertising can help generate revenue and sustain the growth of the chat application.

Overall, the future of friendly chat applications lies in continuous innovation, focusing on enhancing user experience, leveraging emerging technologies, and adapting to evolving user preferences and market trends.

CONCLUSION

Concluding a friendly chat application can be as simple as expressing gratitude for the conversation and wishing the other person well. Here's a sample conclusion:

"It's been great chatting with you! Thanks for the delightful conversation. Have a fantastic day/ night ahead! Take care and hope to chat with you again soon!”

"Time flies when you're having fun! I've really enjoyed our chat—it's been a wonderful exchange of thoughts and ideas. Let's keep this positive energy going! Remember, if you ever need someone to talk to or just want to share something exciting, I'm here. Take care of yourself and cherish the moments ahead. Until next time, stay awesome!”

As our conversation comes to a close, I want to express my appreciation for the connection we've shared. It's been enriching and delightful, filled with laughter, insights, and genuine camaraderie. Let's carry this warmth forward, nurturing our bond and spreading positivity wherever we go. Life is an adventure, and it's made even more meaningful by the connections we forge along the way. So here's to more moments like this—full of friendship, understanding, and joy. Until we meet again, take care, stay safe, and may happiness accompany you on your journey."

REFERENCE

1. Döring, N., & Pöschl, S. (2019). How do messaging apps affect the intensity and satisfaction of social ties? An empirical investigation. Journal of Computer-Mediated Communication.

2. Jin, S. A., & Park, N. (2019). The impact of mobile messaging apps on intergenerational communication: The case of WeChat in China. Mobile Media & Communication.

3. Liu, H., & Liang, J. (2019). Multimedia sharing in instant messaging apps: Effects on social presence, enjoyment, and relationship closeness. Computers in Human Behavior.

4. Hu, Y., Manikonda, L., & Kambhampati, S. (2017). What we Instagram: A first analysis of Instagram photo content and user types. Proceedings of the 8th International Conference on Social Media & Society.

5. Xu, Y., & Teo, H. H. (2019). Examining users' privacy concerns and trust in WeChat: The moderating role of trust propensity. Information Systems Journal.

6. Liao, Q. V., & Fu, W. W. (2017). Protecting user privacy: An analysis of instant messaging applications. Computers in Human Behavior.

7. Chang, C., & Chuang, S. S. (2020). How messaging app usage influences multitasking behaviors: An examination of WeChat users. Telematics and Informatics.

8. Chua, T. H., & Chang, L. (2019). The effects of using messaging apps on users' self-esteem and loneliness. Telematics and Informatics.

9. Kim, J. H., & Kim, J. (2020). AI-based chatbots in messaging apps: Exploring user perceptions and impacts on communication experiences. Journal of Interactive Advertising, 20(1). 4.4 Features of the project Online Blogging System