WeSafe : Enhancing Human Security Mobile App

Project Team

Sl. No.	Reg. No.	Student Name
1	21MPBS408008	Garima Swami
2	21MPBS408003	Bindu Shree P
3	21MPBS408004	Kalash Nag

B. Sc (Hons). in Computer Science

FACULTY OF MATHEMATICS AND PHYSICAL SCIENCES

M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Bengaluru -560 054

FACULTY OF MATHEMATICS AND PHYSICAL SCIENCES

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Certificate

This is to certify that the Group Project titled "WeSafe: Enhancing Human Security Mobile App" is a bonafide record of the project work carriedout in the Department of Computer Sciences by Ms. Garima Swami bearing Reg. No. 21MPBS408008, 2021- 2024 batch in partial fulfilment of requirements for the award of B.Sc.(Hons.) Degree in Computer Science of M. S. Ramaiah University of Applied Sciences.

December - 2023

Dr. Jyothi A P

Assistant Professor, Dept. of CSE

Dr. Chandan Kumar S
Head, Department – CS

Dr. J V Desai
Dean, Faculty of FMPS

FACULTY OF MATHEMATICS AND PHYSICAL SCIENCES

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Certificate

This is to certify that the Group Project titled "WeSafe: Enhancing Human Security Mobile App" is a bonafide record of the project work carriedout in the Department of Computer Sciences by Ms. Bindu Shree P bearing Reg. No. 21MPBS408003, 2021- 2024 batch in partial fulfilment of requirements for the award of B.Sc.(Hons.) Degree in Computer Science of M. S. Ramaiah University of Applied Sciences.

December - 2023

Dr. Jyothi A P

Assistant Professor, Dept. of CSE

Dr. Chandan Kumar S
Head, Department – CS

Dr. J V Desai
Dean, Faculty of FMPS

FACULTY OF MATHEMATICS AND PHYSICAL SCIENCES

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Certificate

This is to certify that the Group Project titled "WeSafe: Enhancing Human Security Mobile App" is a bonafide record of the project work carriedout in the Department of Computer Sciences by Ms. Kalash Nag bearing Reg. No. 21MPBS408004, 2021- 2024 batch in partial fulfilment of requirements for the award of B.Sc.(Hons.) Degree in Computer Science of M. S. Ramaiah University of Applied Sciences.

December - 2023

Dr. Jyothi A P

Assistant Professor, Dept. of CSE

Dr. Chandan Kumar S
Head, Department – CS

Dr. J V Desai
Dean, Faculty of FMPS

Declaration

WeSafe: Enhancing Human Security Mobile App

The project work is submitted in partial fulfilment of academic requirements for the award of B.Sc. (Hons.) Degree in the Department of Computer Science of the Faculty of Mathematics and Physical Sciences of Ramaiah University of Applied Sciences. The project report submitted herewith is a result of our work and in conformance with the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

Sl. No.	Reg. No.	Student Name	Signature
1	21MPBS408008	Garima Swami	
2	21MPBS408003	Bindu Shree P	
3	21MPBS408004	Kalash Nag	

Date: **27 December, 2023**

Acknowledgements

We are pleased to present the **WeSafe: Enhancing Human Security Mobile App** project and take this opportunity to express our profound gratitude to all those people who helped us in the completion of this project.

We thank our college for providing us with excellent facilities that helped us to complete and present this project. We would also like to thank Dean of CSE, Dr. J.V.Desai the staff members and lab assistants for permitting us to use computers in the lab as and when required.

We express our deepest gratitude to our project guide Dr. Jyothi A P, Assistant Professor of CSE for her support and valuable and timely advice during the various phases of our project. We would also like to thank him/her for providing us with all the proper facilities and support as the project co-coordinator. We would like to thank him/her for support, patience and faith in our capabilities and for giving us flexibility in terms of working and reporting schedules.

We would like to thank all our friends for their smiles and friendship making college life enjoyable and memorable and for family members who always stood beside us and provided the utmost important moral support. Finally, we would like to thank everyone who has helped us directly or indirectly with our project.

Abstract

"WeSafe: Enhancing Human Security" is a revolutionary mobile application designed to address the contemporary demand for safety and security. Developed on the Flutter framework, this cross-platform application prioritizes the well-being of women, men, senior citizens, and anyone requiring urgent assistance.

The comprehensive feature set encompasses both front-end and back-end technologies, employing Dart for the front-end and SQLite for the back-end, with Android Studio as the chosen Integrated Development Environment (IDE). The Flutter-based Stay Safe Security System empowers users to connect with their family and friends, allowing them to share their current location seamlessly.

Key User-Centric Features include customizable Scream Alarms, Fake Call Timer for simulated calls, Track Me functionality for location sharing, a Friends List for quick access to contacts, SOS alerts triggered manually or through device shaking, Audio Recording for emergency messages, and customizable language preferences through Settings.

The application aims to overcome the limitations of traditional safety techniques, offering a high-performance solution that ensures accuracy and reliability in emergency scenarios. By addressing the drawbacks of existing systems, such as difficulties in tracking and the lack of alternative emergency options, WeSafe stands as a significant advancement in mobile safety applications. The integration of advanced features and a user-friendly interface makes WeSafe a powerful tool for individuals seeking enhanced security in various situations.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Introduction

The burgeoning enterprises over particular safety, particularly for women but can be used by gender, have prompted the induction of the" WeSafe" design. The World Health Organization(WHO) and the National Crime Records Bureau(NCRB) emphasize the unsettling fact that 35 of women worldwide endure unethical physical importunity. This pervasive issue is not confined to a specific region but manifests itself in public places analogous as road and machine daises, paths, and various cooperative spaces. design explanation The provocation behind bearing the WeSafe design is bedded in the fundamental principle that safety is an introductory demand for every human being.

The caregiving rates of physical importunity, especially against women, illuminate a critical need for a visionary security system that addresses safety enterprises in public places and during single trips on public transport.

Major enterprises taking this as a design are
1. Addressing a Critical Social Issue- WeSafe tackles the pressing issue of physical importunity, offering a comprehensive result that combines technological invention with the imperative need to produce safer public spaces.

2. Promoting Gender Equality- Women's security has become a major societal concern, and WeSafe aims to contribute to the larger discourse on gender equality by furnishing women with tools to navigate public spaces with confidence.

3. icing Personal Safety- Safety is a fundamental moral right, and WeSafe is driven by the belief that everyone should be able to move freely without the constant fear of importunity. This design seeks to empower individuals to take control of their safety. The WeSafe design is not just a technological shot; it's a commitment to icing the fundamental right to particular safety for every existent, regardless of gender or circumstance. By addressing a critical social issue, promoting gender equality, and employing technology for good, WeSafe stands as a beacon for a safer and more secure future.

1.2 Literature Survey

Table 1 Literature survey

s.no.	Author	journal name and year of publication	research focus	finding in research	conclusion derived via authors	limitations in the study	conclusion of published work
1.	Erich H. Fruchtnicht, Leslie D. Lutz, John W. Fellers, Clay D. Hanks	Proactive Safety: Designing & Implementing a Mobile Application. Professional Safety, Vol. 59, No. 8 (AUGUST 2014), pp. 37-42 (6 pages)	The mobile centric nature of the campus user population called for the safety and security information to be presented by a mobile-friendly method.	The mobile app created was of a great help for the campuses for the safety and security of everyone	The app achieves goals such as managing safety information, presenting consistent campus-specific details, and encouraging frequent user interaction with daily-use information. By incorporating feedback and collaborating with TAMUS safety	Dependencies on User Feedbacks. If the app solely depends on used feedbacks, there might be issues which may not be identified	The app was created for better security and safety of campus people it works and provides the goal it was intended but only focuses on user of input and has limitations if used in different campuses

					departments, the HSC EHS team strives to empower users in handling diverse situations.		
2	Wasim Akram, Mohit Jain, C. Sweetlin Hemalatha	Design of a Smart Safety Device for Women using IoT. Procedia Computer Science Volume 165, 2019, Pages 656-662	Providing a safety device for women in critical situation	The device made provided the intended help for women with experimenting with new technical features.	features such as fingerprint recognition, a distress-triggered buzzer, and text message alerts to ensure effective security in stressful situations. The inclusion of a shockwave generator for self-defense adds a practical layer of protection	The usage of certain self-defense technologies, like the shockwave generator, may raise legal and ethical questions, necessitating adherence to regulations and ethical standards to prevent misuse or harm	The proposed project integrates new feature like fingerprint based identification system.
3	B.Sindhu Bala1, M.Swetha2,M.Ta	Survey On Women	Development and integratio	Various technological	The current analysis	The limitations	As technology

	milarasi3 and D.Vinodha4)[4]	Safety Using IOT	n of various technological system aimed at ensuring women safety	advancements have been introduced to address women's safety concerns.	suggests limitations in GPS, GSM, and sensor-based systems, primarily restricted to tracking nearby locations and sending alerts to limited contacts. To address this, a new automated safety system is proposed, aiming to enhance accuracy by detecting multiple physical parameters without human intervention, allowing for immediate alert messages	identified during the survey are limited tracking range, alert reach is limited too, high dependency on user actions and many technical restrains.	advances more innovations are used in women's safety measures

					during potential violations against women.		
4	Kunal Kataria 1 Rushikesh Khade 2 Rohit Kurhade 3 Amit Pende 4 Prof. Sonal Chanderi 5	A Survey Paper on Android App for Women Safety	The main aim of the project is to develop a user-friendly, reliable, and comprehensive safety application that empowers women to seek assistance efficiently during emergency situations.	The findings from this project could encompass user feedback, usability testing results, potential improvements for enhanced functionality, and insights into the practicality and effectiveness of the application in real-world emergency scenarios.	This article discusses the creation of the Android app "Security Alert" for women's protection, Future enhancements could integrate the app with law enforcement databases, potentially offering significant aid during emergencies when the device lacks network access. Overall, the app aims to provide crucial	While the project suggests integration with law enforcement databases as a future enhancement, the current version may lack direct connectivity to emergency services or official databases, potentially impacting the app's response time and effectiveness in critical situations.	Many human security apps are coming into the market with much better version than the existing one.

					assistance in perilous situations for both women and men.		

1.3 Conclusion

In this chapter, we discussed about four papers we had chosen to research and survey for our application. We read through their finding and the issues they add while working on the recommendation systems. We also looked at other similar applications in the same field and see how and what we offer would work better. Keeping in mind these findings and issues, we went on to make our implementation of the recommendation system.

2. Background Theory

2.1 Background Theory

2.1.1 Introduction

The elaboration of mobile operations has unnaturally converted the way individualities interact with technology, bridging the gap between physical and digital realms. The foundation of mobile operations lies in the confluence of colourful technologies, including mobile operating systems, development fabrics, and pall computing. Mobile operations are software programs designed to run on mobile bias, similar to smartphones and tablets. The theoretical underpinnings of mobile app development encompass several crucial factors 1. Mobile Operating Systems- Mobile operations are developed for specific operating systems, similar to Android or iOS. Understanding the complications of these operating systems is vital for creating operations that seamlessly integrate with the device's functionalities. 2. stoner Interface(UI) and stoner Experience(UX) Design- The proposition of UI/ UX design plays a pivotal part in mobile app development. It focuses on creating intuitive interfaces that enhance stoner engagement and satisfaction. Principles similar to responsiveness, availability, and aesthetics contribute to a positive stoner experience. 3. Mobile App Development Frameworks- Frameworks like Flutter give a theoretical frame for erecting cross-platform operations. These fabrics streamline the development process by enabling law reusability across different platforms. 4. Emerging Technologies-Theoretical disquisition of arising technologies, similar to stoked reality(AR), virtual reality(VR), and artificial intelligence(AI), provides perceptivity into creating innovative and slice-edge mobile operations. Understanding these theoretical aspects is essential for inventors and stakeholders to produce mobile operations that aren't only functional but also stoner-friendly, secure, and adaptable to the dynamic geography of technology.

2.1.2 Highlights of the application in comparison to similar applications

In the geography of safety and security operations, WeSafe emerges as a trailblazer, reconsidering the paradigm of particular safety with a host of innovative features and a stoner-centric design that sets it piecemeal from being druthers Unlike numerous operations that offer insulated functionalities, WeSafe stands out by presenting a holistic approach to safety, icing a comprehensive and stoner-friendly experience. One of the crucial differentiators of WeSafe lies in its application of slice-edge real-time position tracking technology. This point transcends the capabilities of operations by furnishing druggies with immediate and accurate updates on their whereabouts. In discrepancy to traditional safety apps, which may suffer from detainments or inaccuracies, WeSafe ensures that help arrives instantly in extremities. also, WeSafe introduces a unique and discreet alert medium through its fake call generator. This point addresses a pivotal gap in operations, allowing

druggies to gesture torture without drawing attention. In comparison, other operations frequently warrant similar covert alert mechanisms, leaving druggies with limited options for seeking backing discreetly. The shareable position link point in WeSafe represents another advance in the realm of safety operations. By enabling druggies to induce a web link associated with their real-time position, WeSafe surpasses being druthers that may warrant precise and immediate backing capabilities. This innovative approach ensures that trusted connections can track druggies on platforms like Google Charts fleetly, easing a more effective and targeted response. A defining aspect of WeSafe is its commitment to a visionary security system that not only responds to extremities but also strives to help them. While numerous living operations concentrate solely on reactive measures, WeSafe integrates a comprehensive set of security measures. This strategic approach positions WeSafe as a forward-allowing result that addresses a diapason of safety scripts. Specialized prowess also contributes to WeSafe's superiority. The operation leverages the Flutter frame, icing cross-platform comity for both Android and iOS bias. This specialized advantage enhances availability, reaching a broader followership compared to operations confined to a single operating system. Similarly, WeSafe's nonstop monitoring and enhancement strategy underscores its fidelity to staying ahead of rising technologies and evolving stoner requirements. Regular updates and advances grounded on stoner feedback distinguish WeSafe from operations that may stagnate in development, icing that druggies profit from the rearmost advancements in safety technology.

2.1.3 Dart Programming Language

Google created Dart, a slice-edge, object-acquainted programming language for erecting online, mobile, and garçon operations. Since its 2011 launch, Dart has gained fashion ability, particularly among inventors of mobile apps that make use of the Flutter frame. Dart offers a strong base for erecting high-performance operations by combining the voguish characteristics of roundly and statically compartmented languages. WeSafe thickness Across Platforms, Productive Development Workflow, Performance Optimisation, and inflexibility to Modern Development practices are Benefits of opting for Dart. WeSafe's front-end development is erected on top of Dart, which provides an important mix of effectiveness, productivity, and performance. WeSafe is well- deposited to impact the rearmost developments and features as the programming language develops and gains community acceptance, adding icing to a solid and slice-edge safety system for addicts.

2.1.4 Flutter

Flutter has surfaced as a transformative force, reconsidering how inventors approach the creation of cross-platform operations. Conceived and nurtured by Google, Flutter is an open-source UI toolkit that has charmed the inventor community since its preface in 2017. At its core, Flutter is more than just a toolkit; it represents a comprehensive frame for

erecting natively collected operations across different platforms, including mobile, web, and desktop. The guiding gospel behind Flutter is the consummation of a single codebase that can seamlessly restate into operations that run on both Android and iOS bias, as well as extend to web cybersurfs and desktop surroundings. Flutter's pledge of" write formerly, run anywhere" isn't just a tagline but a transformative conception that addresses a longstanding challenge in the assiduity. inventors can now produce a singular codebase for their operations, reducing redundancy and streamlining the conservation process. This cross-platform nature has deposited Flutter as a game-changer for businesses seeking effective and cost-effective results for reaching a wide stoner base. The frame also boasts a rich set of designed contraptions that feed to both Material Design, Google's design language, and Cupertino, Apple's iOS design language. This ensures that operations developed with Flutter maintain a native look and feel on both Android and iOS platforms, contributing to a harmonious and familiar stoner experience.

2.1.5 SQLite

The world of mobile app development is, about finding efficient and streamlined solutions to manage data. In Android Studio when using Azure as a database, one powerful and lightweight relational database management system (RDBMS) that stands out is SQLite. It is widely adopted in the mobile development landscape because it aligns perfectly with the needs of apps.

What makes SQLite special is its simplicity and minimalistic design, which makes it an excellent choice for environments with resources. It's a self-contained and serverless architecture that requires configuration making it easy for developers to integrate into their apps without needing a separate database server. This self-contained nature significantly reduces the complexity of managing databases which is an advantage, in the world of mobile development.

One standout feature of SQLite is its "zero configuration" setup. Developers can seamlessly incorporate SQLite into their apps without having to go through any setup or ongoing maintenance tasks. This straightforward deployment process contributes to a hassle-free development experience especially when creating user-friendly mobile apps that are easy to install.

2.1.6 Android Studio

Android Studio Empowering Developers in the Android Ecosystem- In the dynamic geography of mobile operation development, Android Studio stands as a vital tool that has played a transformative part in shaping the Android ecosystem. Launched by Google in 2013, Android Studio is the sanctioned Integrated Development Environment(IDE) for Android app development. Its elaboration over time has imaged the rapid-fire advancements in the Android platform, furnishing inventors with a robust and point-rich terrain to produce innovative and stoner-centric operations. Android Studio embodies a stoner-centric design that prioritizes effectiveness and ease of use. The IDE integrates seamlessly with the Android

operating system, offering a familiar interface for inventors. The well-designed layout, intuitive navigation, and customizable interface contribute to a positive development experience. From the law editor to the visual layout editor, Android Studio provides a cohesive terrain that caters to inventors of varying skill situations.
The IDE supports multiple programming languages, including Java and Kotlin, furnishing inflexibility for inventors to choose their favoured language. Intelligent law backing, real-time error checking, and bus-completion features streamline the coding process, reducing the liability of syntax crimes and enhancing the overall quality of the codebase.

2.1.7 Kotlin

Kotlin Transforming Android Development with Elegance and Innovation In the ever-evolving geography of Android app development, Kotlin has surfaced as a language of choice, reconsidering the way inventors craft operations for the world's most popular mobile operating system. Introduced by JetBrains in 2011 and latterly championed by Google as a sanctioned language for Android development in 2017, Kotlin brings a mix of fineness, expressiveness, and ultramodern features that have propelled it to the van of Android programming. The Rise of Kotlin The relinquishment of Kotlin in the Android ecosystem has been nothing short of a paradigm shift. As a statically- compartmented programming language that runs on the Java Virtual Machine(JVM), Kotlin offers flawless interoperability with Java codebases, making it a seductive option for inventors looking to enhance their Android systems. Its terse syntax, suggestive language features, and strong null safety have addressed numerous pain points associated with Java, furnishing inventors with a more pleasurable and productive development experience. Conciseness and Readability One of Kotlin's hallmark features is its terse syntax, which allows inventors to express complex ideas with minimum boilerplate law. The reduction in verbalism not only accelerates the development process but also improves law readability. Kotlin's suggestive nature encourages inventors to concentrate on the sense of their operations rather than scuffling with gratuitous syntax, leading to cleaner and further justifiable codebases.

2.2 Technological Stack and Dependencies

Flutter by Google is chosen as it will allow us to create natively compiled, fast and beautiful applications for both Android and IOS from a single code base.

SQLite is used as our database to create, update and access data in real-time, and move the overhead of maintaining the database from the application to SQLite.

2.3 Merits and Demerits

Merits:

- Users can easily access the application for help in an emergency
- Application is user-friendly

- Alert System: The application offers scream alert options for both males and females to seek help and deter potential threats.
- The application has an option of generating a web link that enables the real-time location of the user.
- Users can add friends and family with their details ensuring quick access to vital contacts and facilitating efficient communication during emergencies.
- SOS Feature with automated alerts, activated by shaking the phone and triggers the automated alerts via emails and messages to the user's predefined contacts.
- Audio recordings in emergencies are enabled and can be sent.
- Personalization with language settings

Demerits:

- The device will need to be actively on to use this system.

Applicability:

• The interface of the application is user-friendly.

• It helps the user to quickly decide what they want to eat.

2.4 Conclusion

The Background proposition for this operation has been explained in detail. The highlights of the operation in comparison to other operations have been specified. The technological mound and dependences for this operation were listed out. The advantages, and disadvantages.

3. Aim and Objectives

3.1 Title

WeSafe: Enhancing Human Security Application

3.2 Aim

To design and develop a Flutter-based application for Enhancing Human Security.

3.3 Objectives

The objectives of the proposed project are listed below:

1. To conduct a literature survey on the existing Human Security applications, Flutter and its methods and methodologies.
2. To formulate the requirements for the implementation based on the survey.
3. To design the application based on the identified requirements.
4. To implement the design using Flutter, Dart and SQLite
5. To test and validate the developed application.
6. To document the project report based on the university template.

3.4 Methods and Methodology

Table 2 Methods and Methodology

Objective No.	Statement of the Objective	Method/ Methodology	Resources Utilised
1	To conduct a literature survey on the existing Human Security applications,	1.1 Literature review on App development , Flutter and referring to related published papers with	Flutter documentation and YouTube Videos.

	Flutter and its methods and methodologies.	genuine citations and similar existing applications. 1.2 Using this literature research, the necessary requirements would be collected and documented.	
2	To formulate the requirements for the implementation based on the survey.	2.1 Functional and non-functional requirements are derived along with their dependencies.	Software development fundamentals textbooks and our guide.
3	To design the application based on the identified requirements.	3.1 The functional requirements will be used to create high level UML diagrams such as use case, class, activity and sequence diagrams. 3.2 To detail the design, a low level diagram (Flowchart) will be used.	DIA
4	To implement the design using Flutter, dart and SQLite.	4.1 To design the application using Flutter, dart and SQLite. 4.2 The Application will be developed on Android Studio.	Flutter documentations and YouTube videos.
5	To test and validate the developed application.	5.1 Test the application in accordance with requirements.	Flutter tools.
6	To document the project report based on university template.	6.1 Develop a technical report based on the specifications. 6.2 Demonstrate the application before the panel.	Word for documents, powerpoint presentations and Internet resources.

3.5 Conclusion

This particular chapter focused on defining the title and Aim of the project correctly. Later this chapter included the required objectives that were required to be fulfilled to complete this project. Method and methodologies are documented successfully in this section and is successfully tabulated in order to know the steps used in completing the objectives including resources used.

4. Problem Solving

4.1 Project Concept

- The application is based on emergency alerts, fake call timers, audio recordings, Live location tracking, SOS, and adding family and friends for emergency alerts.
- Users can reach out to their family and friends immediately.

Features:

1. Users can easily access the application for help in an emergency
2. Application is user-friendly
3. Alert System: The application offers scream alert options for both males and females to seek help and deter potential threats.
4. The application has an option of generating a web link that enables the real-time location of the user.
5. Users can add friends and family with their details ensuring quick access to vital contacts and facilitating efficient communication during emergencies.
6. SOS Feature with automated alerts, activated by shaking the phone and triggers the automated alerts via emails and messages to the user's predefined contacts.
7. Audio recordings in emergencies are enabled and can be sent.
8. Personalization with language settings

4.2 Design

- **Functional and Non-Functional requirements**
- **Diagrams**
  - Use Case Diagram
  - E-R Diagram
  - Sequence Diagram
  - Activity Diagram
  - Class Diagram
  - DFD Diagrams

4.2.1 Functional Requirements

FR 1. The Users must be able to sign up and sign in and provide essential information for identification.

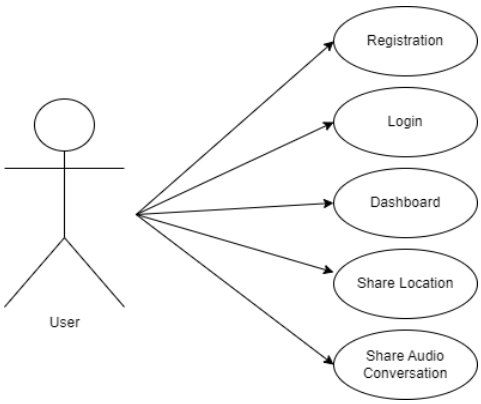
FR 2. The user can use emergency voice screams like male or female scream alarms as well as police sirens so that when they are in danger the alarms will scream and the user can reach out to the people who can help them to stay safe.

FR 3. The user should be able to share their current location with their family or friends who can reach out to them by tracking their live location.
FR 4. The user should be able to share the web link (generated when the user turns their shared location on) with anyone who can help them by tracking their location.
FR 5. The user should be able to make a fake call and the call will be initiated for a minute with the caller's name and number.
FR 6. The user should be able to access the list of family members and friends which will display their details for quick access.
FR 7. The user should be able to turn on the SOS feature (when in an emergency, they shake their mobile phone five times an email or SMS will be sent to all friends list members with a location link).
FR 8. The user should be able to access a predefined default number, typically designated for emergency services such as the Police Station, which is seamlessly integrated into the system to automatically receive the same distress message.
FR 9. The users should be able to record an audio message in a critical situation, and this recording can be accessed by their friends through a dedicated web page generated by the system.
FR 10. The user must be able to choose their favourite contacts for an emergency number.
FR 11. The user must be allowed to set the settings module to personalize their emergency messages by choosing the preferred language for communication.

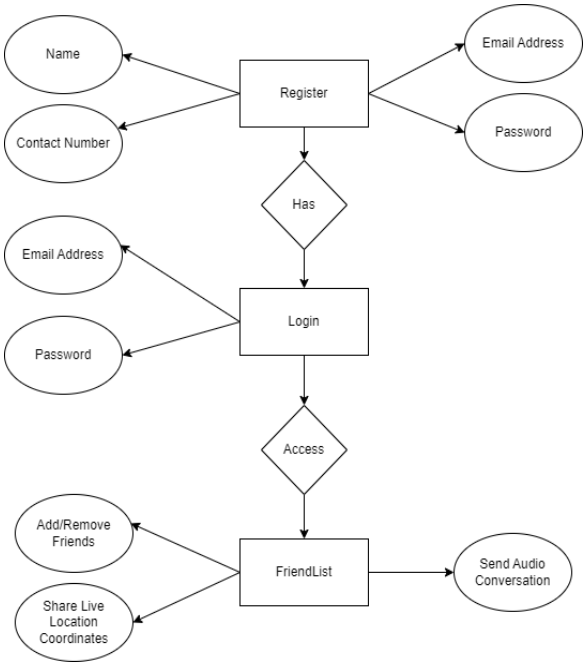
4.2.2 Non-Functional Requirements:

NFR1 Platform independent: The application shall work on all platforms that Flutter supports.
NFR2 Reliability: The application will retain all data to be sent even in no-network situations and send them whenever a network is established.

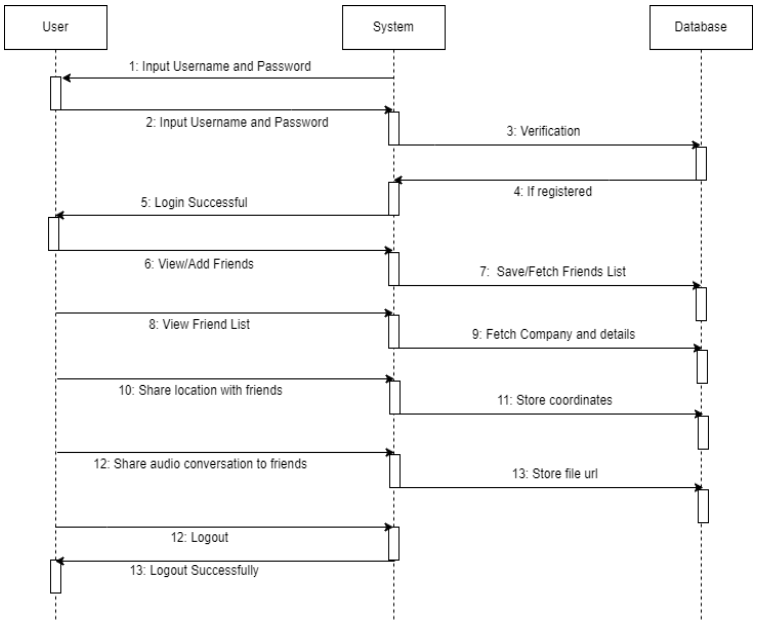
4.2.3 Use Case Diagram



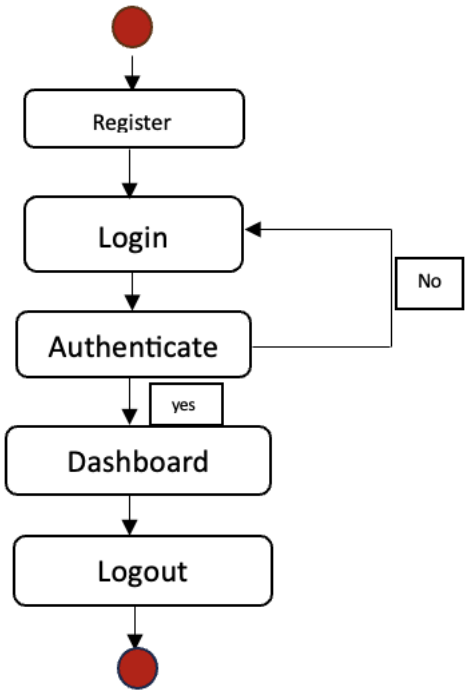
4.2.4 E-R Diagram



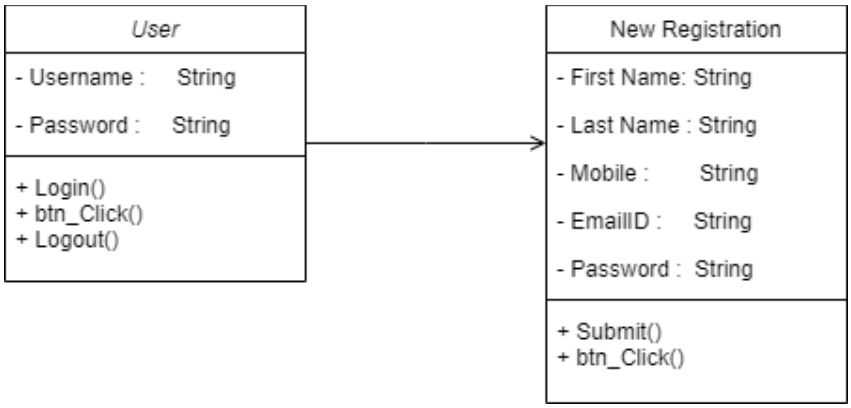
4.2.5 Sequence Diagram



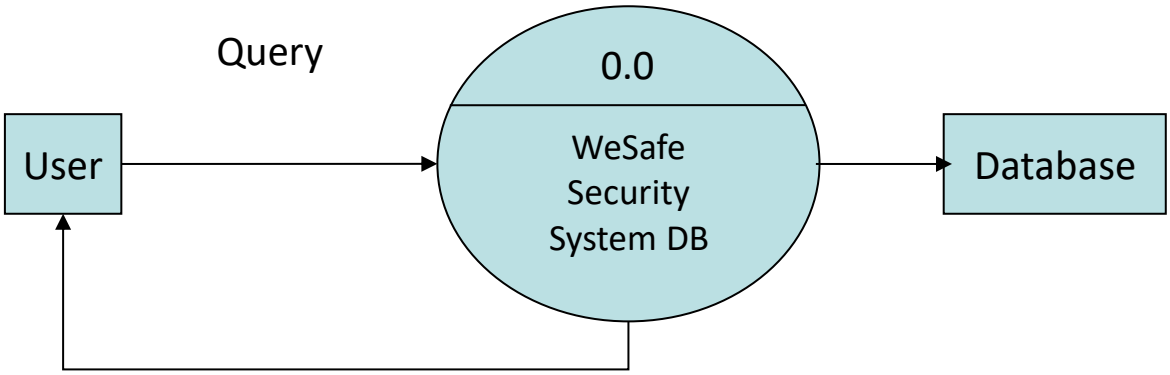
4.2.6 Activity Diagram



4.2.7 Class Diagram

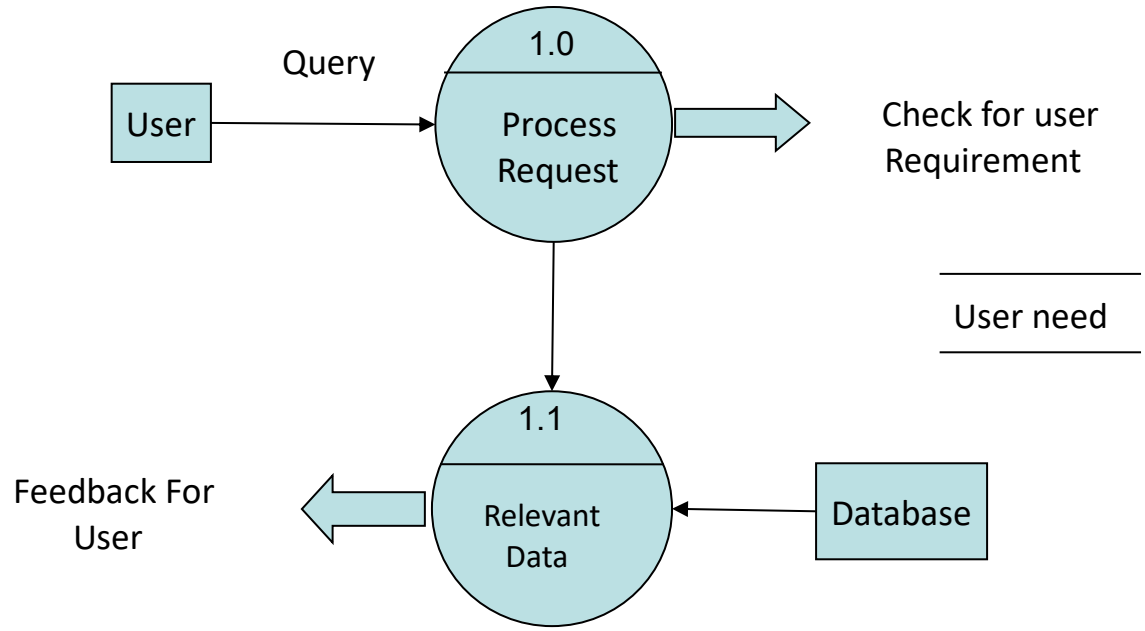


4.2.8 Data Flow Diagram

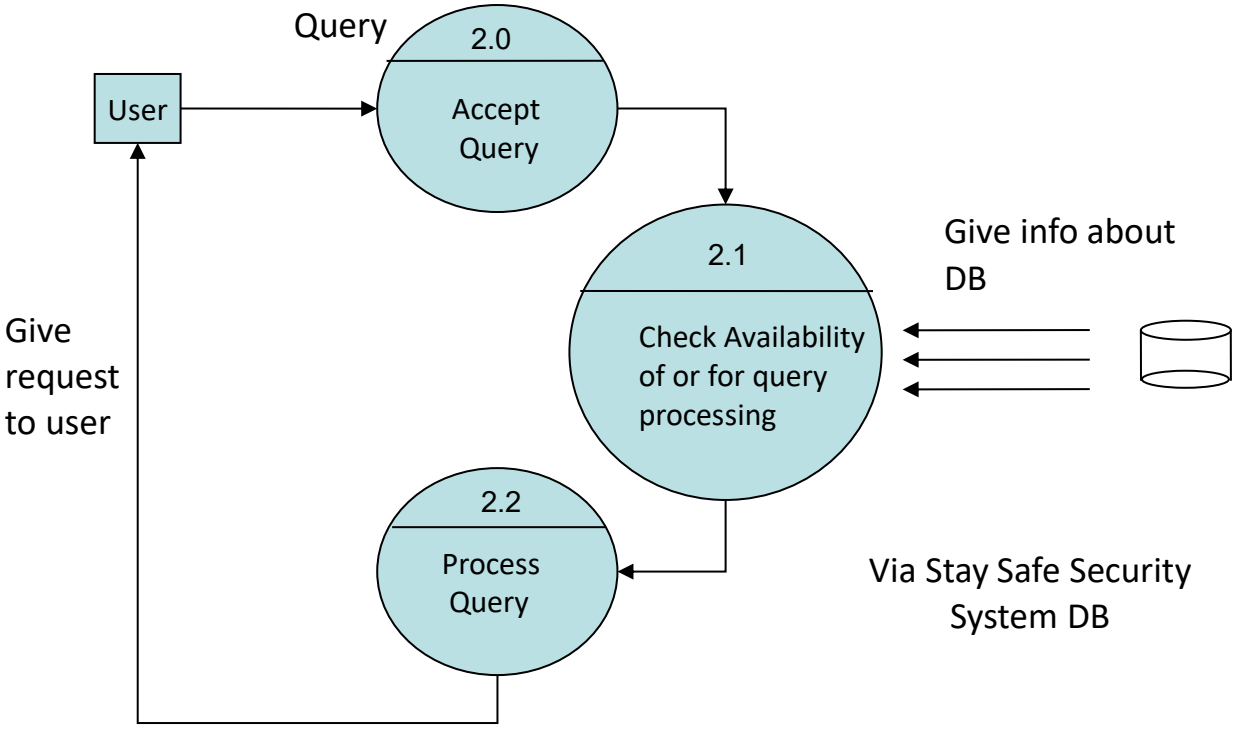


DATABASE DETAIL

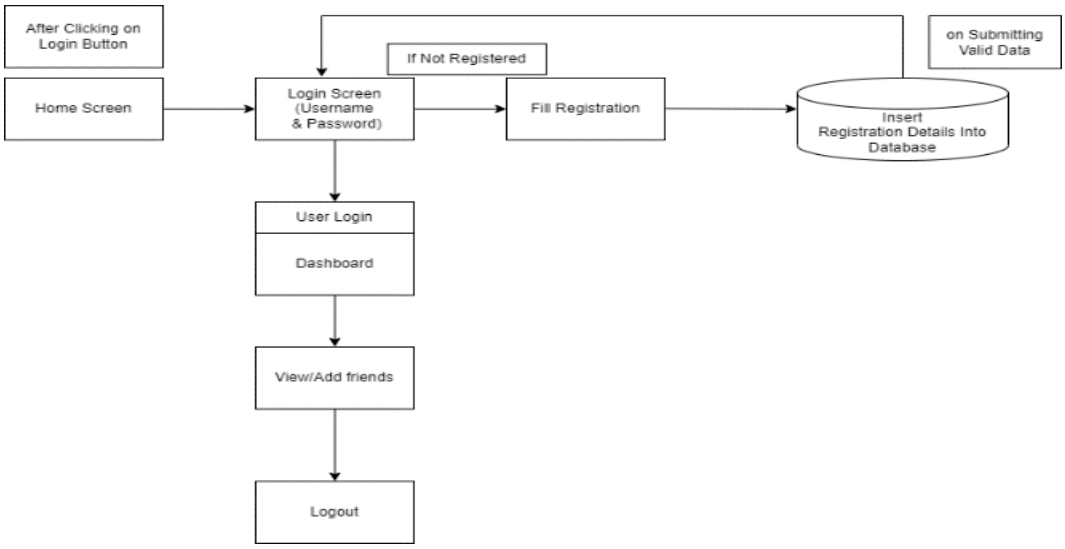
LEVEL 1 DFD


1.0
Query
User
Process
Request
Check for user
Requirement
User need
1.1
Feedback For
User
Relevant
Data
Database

LEVEL 2 DFD: PREDICTION



4.2.9 System Architecture



4.3 Implementation

4.3.1 Project Implementation Technology

The Project application is implemented in Android Studio. We used Android Studio for the Design and coding of the project. Created and maintained all databases in SQL Server, in that we create tables, and write queries for storing data or records of projects.

I. <u>Hardware Requirement</u>

i. Laptop or PC

- MacOS Sierra and above
- Windows 7 or higher
- I3 processor system or higher
- 8 GB RAM or higher
- 100 GB ROM or higher

ii. Android Phone (6.0 and above)

iii. iPhone (iOS 9 and above) (if iOS version needs to be checked)

II. Software Requirement

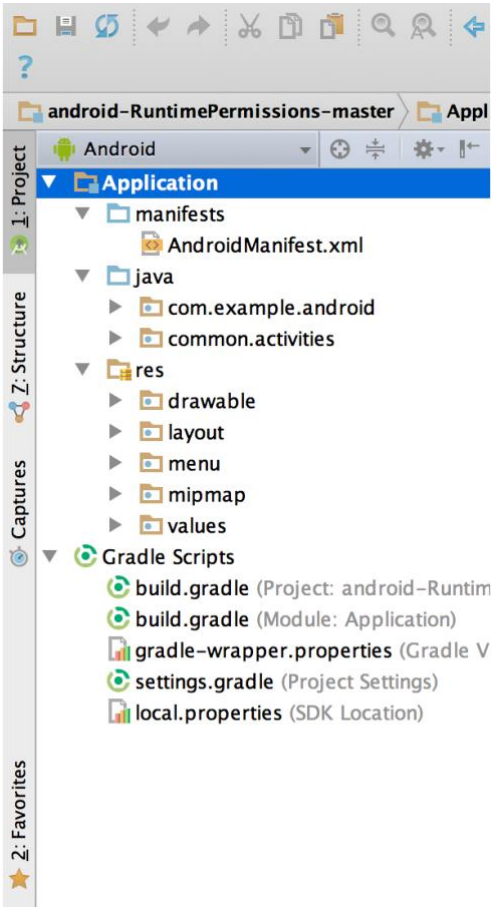
i. Laptop or PC

- Android Studio with Flutter Plugin
- XCode (Latest version) (If iOS version needs to be checked on Mac)
- Azure Data Studio

4.4 Project Structure

Every project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules



By default, Android Studio displays project files in the Android project view, as shown in Figure 1. This view organizes the modules to provide the user quick access to their project's key source files.

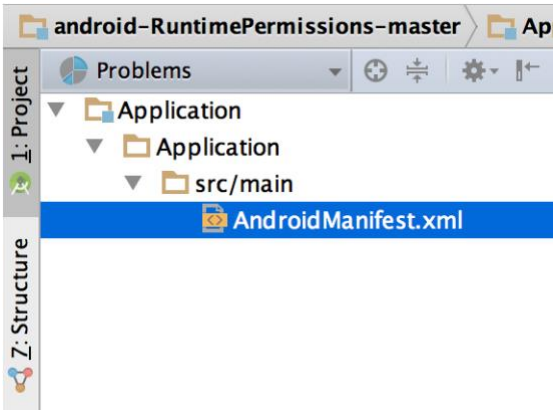
All the build files are visible under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: It Contains the Java source code files, which also include JUnit test code.

Figure 1

- **res**: It contains all the non-code resources, similar to XML layouts, UI strings, and bitmap images.

The Android project structure on disk is different from the provided flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in Figure 1, it's showing android).



We can also customize the view of the project files according to convenience to keep focus on specific aspects of your application development. For instance, selecting the **Problem** view of project shows links to the source files containing any recognized coding and syntax errors,

such as a missing XML element closing tag in a layout file.

4.4.1 The User Interface

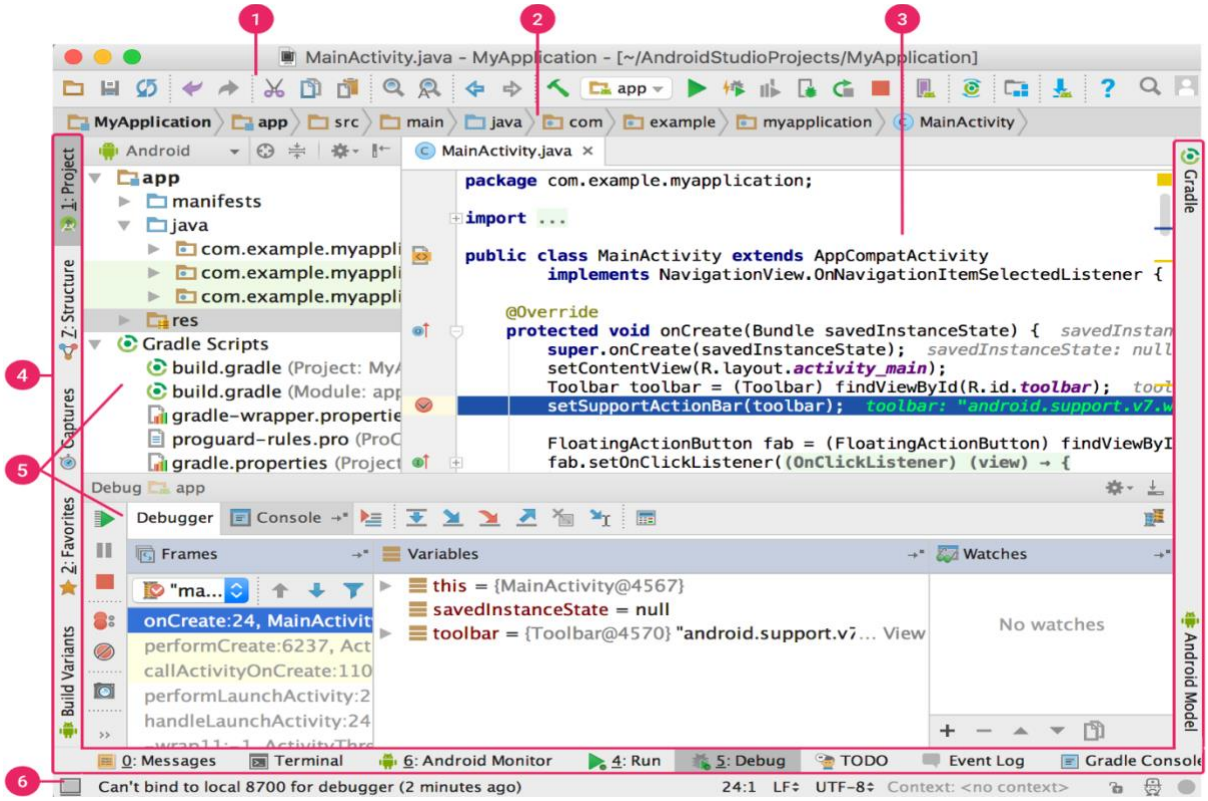
1. The **toolbar** allows you to carry out a wide range of actions, including running your application and launching more Android tools.
2. The **navigation bar** assists you in navigating through the project and also opens files for editing. It provides a more compact view of the structure that is visible in the **Project** window.
3. The **editor window** is where we can create and modify code. Depending on the current file type, the editor can make changes that are required. For instance, when we view a layout file, the editor shows the Layout Editor.

4. The **tool window bar** runs around the outside of the Integrated Development Environment window and contains the buttons that allow you to expand or collapse individual tool windows.

5. The **tool windows** allows you to specific tasks like project management, search, version control, and more. You can also expand them and collapse them.

6. The **tool windows** allow access to specific tasks like project management, search, version control, and more. We can expand them and collapse them.

7. The **status bar** displays the status of the project and the IDE itself, as well as any warnings or messages.



We can organize the main window to give more screen space by hiding or moving toolbars and tool windows. We can also use keyboard shortcuts to access most Integrated Development Environment features.

At any time, we can also search across the source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key or clicking on the magnifying glass icon in the upper right corner of the Android Studio window. This can be very useful if, for instance, we are trying to locate a particular IDE action that we have forgotten how to trigger.

4.4.2 Tool Windows

Instead of using pre-set perspectives, Android Studio follows context and automatically brings up relevant tool windows as the user starts working. By default, the mostly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click on the tool's name icon in the tool window bar. We can also drag, pin, unpin, attach, and detach tool windows.
- To go back to the current default tool window layout, click **Window > Restore Default Layout** or customize default layout by clicking **Window > Store Current Layout as Default**.
- To show or hide the entire tool window bar, click the window icon  in the bottom left-hand corner of the Android Studio window.
- To find a specific tool window, hang over the window icon and select the tool window from the menu.

4.4.3 Navigation

Here are some tips to help you move around Android Studio.

- Switch between recently accessed files using the *Recent Files* action. Press **Control+E** (**Command+E** on a Mac) to bring up the Recent Files action. By default, the previously accessed file is selected. We can also access any tool window through the left column in this action.
- Look at the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12** (**Command+F12** on a Mac). Using this action, we can quickly navigate to any part of the current file.
- Search for and navigate to a specific class in a project using the *Navigate to Class* action. Fetch up the action by pressing **Control+N**(**Command+O** on a Mac). Navigate to Class supports sophisticated expressions, including camel humps, paths, line navigator, middle name matching, and many more. If we call it twice in a row, it shows the results of the project classes.
- Use *Navigate to File* action to navigate to a file or folder. Take up the Navigate to File action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac).
- Navigate a method or field by using the *Navigate to Symbol* action. Bring up the Navigate to Symbol action by pressing **Control+Shift+Alt+N**(**Command+Shift+Alt+O** on a Mac).
- Observe all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7**

4.4.4 Gradle Build System

The foundation of the build system used for Gradle is Android Studio, with more Android-specific capabilities provided by the Android plugin for Gradle. We can use the features of the build system to do the following:

- It also customizes, configures, and extends the build process.
- It creates multiple APKs for applications, with different features using the same project and modules.
- It reuses the code and the resources across source sets.

Because of the flexibility of Gradle, we can achieve all of this without modifying your app's core source files. Android Studio build files are named as build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Every project has one top-level build file for the complete project and separate module-level build files for each module. When we import an existing project, Android Studio automatically generates the necessary build files.

4.4.5 Performance monitors

Android Studio provides performance monitors so we can more easily track our app’s memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and examine network requests. With our app running on a device or emulator, open the **Android Monitor tool** window, and then click the **Monitors** tab.

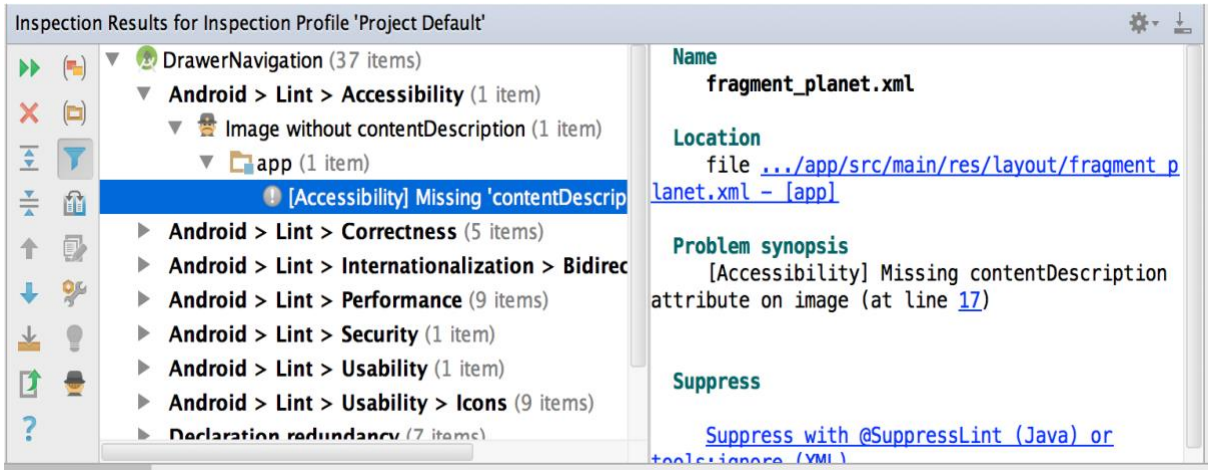
4.4.6 Allocation tracker

Android Studio allows tracking memory allocation as it monitors memory use. Tracking memory allocation allows us to monitor where objects are being allocated when we perform certain actions. Knowing these allocations enables us to optimize our app's performance and memory use by adjusting the method calls related to those actions.

4.4.7 Code inspections

Whenever we compile our program, Android Studio automatically runs configured Lint and other IDE inspections to help us easily identify and correct problems with the structural quality of our code.

The Lint tool checks our Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.



4.4.8 Install the Flutter and Dart plugins:

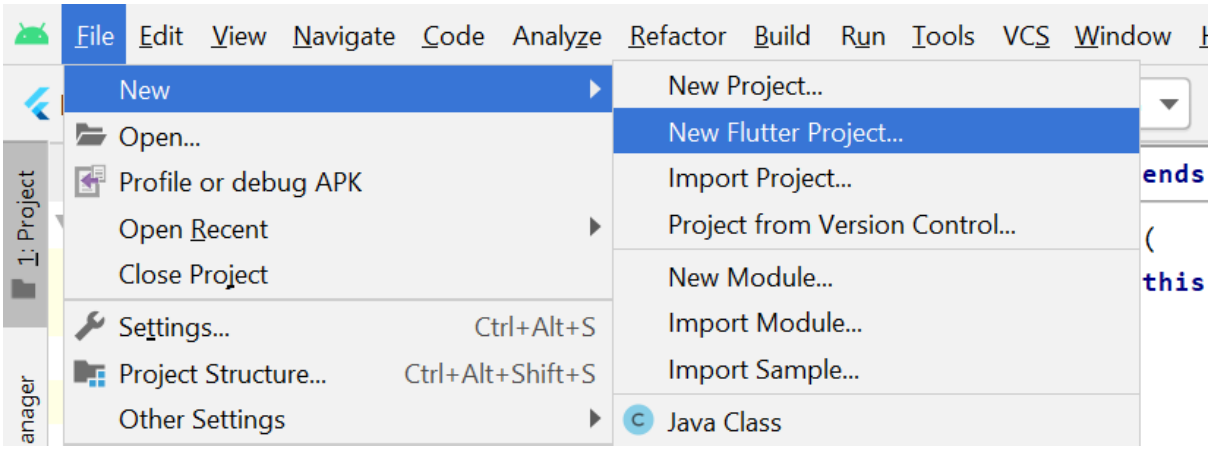
Once the Android Studio is installed, you have to install Flutter and Dart plugins.

1. **Start** Android Studio.
2. Open plugin preferences **(Configure > Plugins as of v3.6.3.0 or later)**.
3. Select the **Flutter** plugin in the dropdown and click on Install.
4. Click Yes when it's prompted to install the **Dart** plugin and follow it.
5. Click **Restart** when prompted.

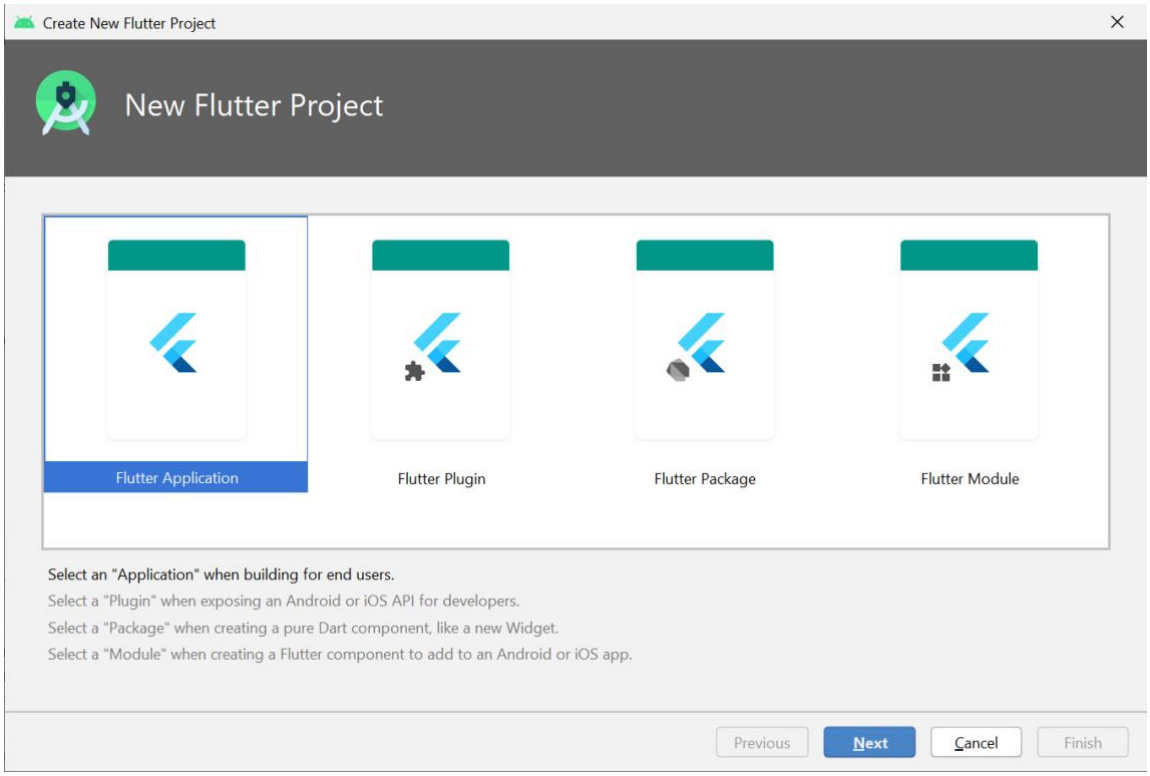
4.4.9 Creating the application:

After installing Dart and Flutter plugins user needs to create a Flutter app to make sure it is working properly or not, for the same follow the steps mentioned below:

Step 1: Open the **Integrated Development Environment** and select Start a **new Flutter project**.



Step 2: Select the **Flutter Application** as your project type. Then click **Next**.



Step 3: Verify the **Flutter Software Development Kit path** specifies the SDK's location **(select Install SDK… if the text field is blank)**.

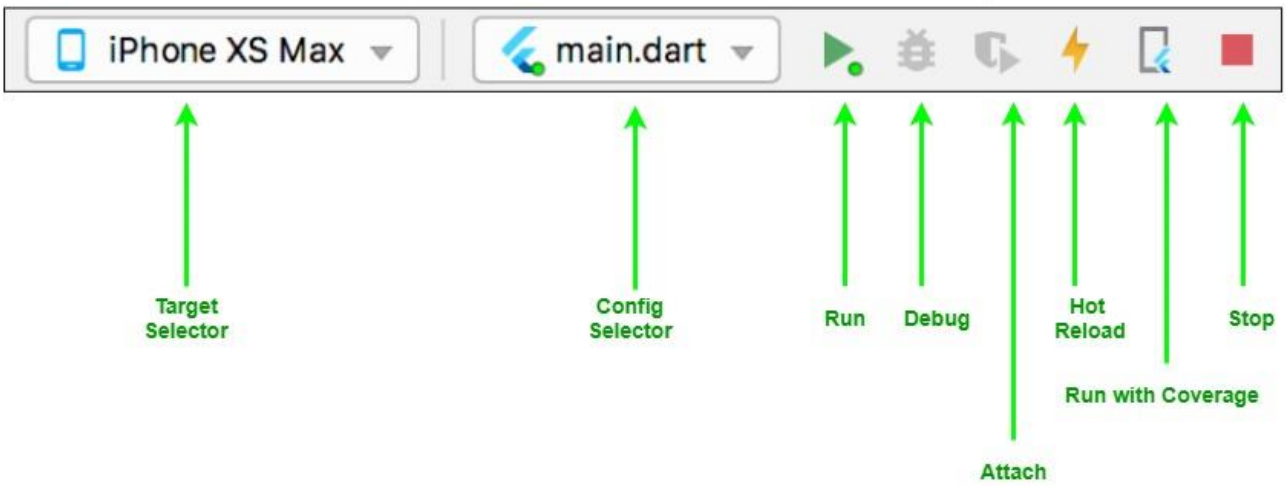
Step 4: Enter your project name (for instance, my app). Then click **Next**.

Step 5: Click **Finish**.

Running the application:

Do Follow the below steps to run the flutter application you made that was structured above:

Step 1: Locate the main Android Studio toolbar:



Step 2: In the **target selector**, select an Android device for running your Flutter application.

Step 3: Click on the run icon in the toolbar, or invoke the menu item **Run > Run**.
After the application build is completed, you'll see the starter app on your device.

5.CODING

5.1 Implementation to Add Friends in the application

```
class AddFriendController extends GetxController {
  late TextEditingController etFriendName;
  late TextEditingController etEmailId;
  late TextEditingController etContactNumber;
  late TextEditingController etFamilyDetails;

  late FocusNode etFriendNameFocusNode;
  late FocusNode etEmailIdFocusNode;
  late FocusNode etContactNumberFocusNode;
  late FocusNode etFamilyDetailsFocusNode;

  RxBool isEditingSavedFriend = false.obs;
  late UserFriends? userFriendsArg;
  late String? userFriendId;
  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  @override
  void onInit() {
    super.onInit();
    initUI();
    initObj();
    setArgumentDetailsToFields();
  }

  void initUI() {
    etFriendName = TextEditingController();
    etEmailId = TextEditingController();
    etContactNumber = TextEditingController();
    etFamilyDetails = TextEditingController();

    etFriendNameFocusNode = FocusNode();
    etEmailIdFocusNode = FocusNode();
    etContactNumberFocusNode = FocusNode();
    etFamilyDetailsFocusNode = FocusNode();
  }

  void initObj() {
    userFriendsArg = Get.arguments;
  }

  void setArgumentDetailsToFields() {
    if (userFriendsArg != null) {
      userFriendId = userFriendsArg?.userFriendId;
      if (userFriendId != null && userFriendId.toString().trim() != "") {
        isEditingSavedFriend.value = true;
      }
      etFriendName.text = userFriendsArg?.friendName ?? "";
      etContactNumber.text = userFriendsArg?.contactNumber ?? "";
      etEmailId.text = userFriendsArg?.emailId ?? "";
      etFamilyDetails.text = userFriendsArg?.familyDetails ?? "";
    }
  }

  void onClickSubmit() {
    formKey.currentState!.validate() ? register() : null;
  }

  void onPressDelete() {
    CommonHelper.deleteDialog(
      toDelete: "friend",
      onConfirmDelete: () => _removeUserFriend(),
```

```
    );
  }

  Future<UserFriends> createUserFriendsObject() async {
    UserFriends userFriend = UserFriends();
    userFriend.userId = await UserPref.getUserId();
    userFriend.friendName = etFriendName.text;
    userFriend.contactNumber = etContactNumber.text;
    userFriend.emailId = etEmailId.text.toString().trim();
    userFriend.familyDetails = etFamilyDetails.text.toString().trim();
    return userFriend;
  }

  Future<UserFriends> createUserFriendsObjectToRemove() async {
    UserFriends userFriend = UserFriends();
    userFriend.userId = await UserPref.getUserId();
    userFriend.userFriendId = userFriendId;
    return userFriend;
  }

  Future<void> register() async {
    try {
      await ApiProvider.postMethod(
        url: ApiConstants.addFriends,
        obj: (await createUserFriendsObject()).toJson(),
      ).then((response) {
        List userFriendsList = response.map((e) {
          return UserFriends.fromJson(e);
        }).toList();
        if (userFriendsList.isNotEmpty) {
          if ((userFriendsList.first.status != null) &&
                  userFriendsList.first.status == 'true' ||
              userFriendsList.first.status == 'ok') {
            onSaveFriendSuccessResponse();
          } else {
            onFailedResponse(userFriendsList.first.status.toString());
          }
        } else {
          onFailedResponse("");
        }
      });
    } catch (e) {
      onFailedResponse(e.toString());
    }
  }

  Future<void> _removeUserFriend() async {
    try {
      await ApiProvider.postMethod(
        url: ApiConstants.removeFriends,
        obj: (await createUserFriendsObjectToRemove()).toJson(),
      ).then((response) {
        List userFriendsList = response.map((e) {
          return UserFriends.fromJson(e);
        }).toList();
        if (userFriendsList.isNotEmpty) {
          if ((userFriendsList.first.status != null) &&
                  userFriendsList.first.status == 'true' ||
              userFriendsList.first.status == 'ok') {
            onRemoveFriendSuccessResponse();
          } else {
```

```
            onFailedResponse(userFriendsList.first.status.toString());
          }
        } else {
          onFailedResponse("");
        }
      });
    } catch (e) {
      onFailedResponse(e.toString());
    }
  }

  void onSaveFriendSuccessResponse() {
    Get.back();
    SnackBarUtils.normalSnackBar(
      title: "Success",
      message: "Friend added successfully",
    );
  }

  void onRemoveFriendSuccessResponse() {
    Get.back();
    SnackBarUtils.normalSnackBar(
      title: "Success",
      message: "Friend removed successfully",
    );
  }

  void onFailedResponse(String response) {
    if (response.toLowerCase().contains('already')) {
      SnackBarUtils.errorSnackBar(
        title: "Failed",
        message: "Friend already exists",
      );
    } else {
      SnackBarUtils.errorSnackBar(
        title: "Failed",
        message: "Something went wrong",
      );
    }
  }
}
```

Figure 5. 1 Code for adding the friends.

Using this implementation of Add Friends, we are able to add our family members and friends according to our choice with their contact number, email ID and details of family members or friends. It uses the inputs taken from the user's screen during the initialization. This information is then used to show up in emergency contacts and send alert messages to the same.

5.2 Implementation for Dashboard Of the application

```
class DashboardController extends GetxController {
  RxBool isAudioPlaying = false.obs;
  RxBool isAudioRecording = false.obs;
  RxBool isSosActivated = false.obs;
  RxBool isTrackMeActivated = false.obs;

  late AudioPlayer audioPlayer;
  late AudioRecorder audioRecorder;
  late String recordingPath;

  @override
  Future<void> onInit() async {
    super.onInit();
    await initObj();
    checkIfSosActivated();
    checkIfTrackMeActivated();
    startService();
  }

  @override
  void onClose() {
    super.onClose();
    audioPlayer.dispose();
    audioRecorder.dispose();
  }

  Future<void> initObj() async {
    audioPlayer = AudioPlayer();
    audioRecorder = AudioRecorder();

    DateFormat dateFormat = DateFormat('yyyyMMddHHmmss');
    String timeStamp = dateFormat.format(DateTime.now());
    Directory directory = await getTemporaryDirectory();
    recordingPath = "${directory.path}/$timeStamp.wav";
  }

  Future<void> startService() async {
    FlutterBackgroundService().invoke("setAsBackground");
  }

  void checkIfTrackMeActivated() async {
    isTrackMeActivated.value = await UserPref.getIsTrackMeActivated();
  }

  void checkIfSosActivated() async {
    isSosActivated.value = await UserPref.getIsSOSActivated();
  }

  void onTapLogoutButton() {
    Get.dialog(
      CommonDialog(
        title: "Logout",
        contentWidget: const Text("Are you sure you want to logout?"),
        negativeRedDialogBtnText: "Confirm",
        positiveDialogBtnText: "Back",
        onNegativeRedBtnClicked: () {
          Get.back();
          logout();
        },
      ),
    );
  }

  Future<void> onTapDashboardCard({required String? text}) async {
    text = text.toString().toLowerCase();

    if (text.contains("scream")) {
      _onTapScreamCard();
    } else if (text.contains("record")) {
      _onTapRecordCard();
    } else if (text.contains("sos")) {
      bool locationPermission = await CommonHelper.checkLocationPermission();
      if (Get.isDialogOpen == false && locationPermission == true) {
        Get.toNamed(RouteConstants.sosScreen)?.then((value) async {
          checkIfSosActivated();
        });
      }
    } else if (text.contains("fake")) {
      Get.toNamed(RouteConstants.fakeCallScreen);
    } else if (text.contains("track")) {
      bool locationPermission = await CommonHelper.checkLocationPermission();
      if (Get.isDialogOpen == false && locationPermission == true) {
        Get.toNamed(RouteConstants.trackMeScreen)?.then((value) async {
          checkIfTrackMeActivated();
        });
      }
    } else if (text.contains("friend")) {
      Get.toNamed(RouteConstants.viewFriendsScreen);
    }
  }

  void _onTapScreamCard() {
    audioPlayer.playing ? _stopAudio() : Get.dialog(_optionDialog());
  }

  Future<void> _onTapRecordCard() async {
    await audioRecorder.isRecording() ? _stopRecording() : startRecording();
  }

  Widget _optionDialog() {
    return SimpleDialog(
      children: [
        SimpleDialogOption(
          onPressed: () => playAudio(asset: StringConstants.maleScream),
          padding: const EdgeInsets.all(DimenConstants.contentPadding),
          child: const Text('Male Voice Scream'),
        ),
        SimpleDialogOption(
          onPressed: () => playAudio(asset: StringConstants.womanScream),
          padding: const EdgeInsets.all(DimenConstants.contentPadding),
          child: const Text('Female Voice Scream'),
        ),
```

```
      SimpleDialogOption(
        onPressed: () => playAudio(asset: StringConstants.policeSiren),
        padding: const EdgeInsets.all(DimenConstants.contentPadding),
        child: const Text('Police Voice Siren'),
      )
    ],
  );
}

Future<void> playAudio({required String asset}) async {
  try {
    if (Get.isDialogOpen == true) Get.back();
    isAudioPlaying.value = true;
    audioPlayer.setAsset(asset, preload: true);
    audioPlayer.play();
    audioPlayer.setLoopMode(LoopMode.one);
  } catch (e) {
    CommonHelper.printDebug(e);
    isAudioPlaying.value = false;
  }
}

void _stopAudio() {
  try {
    isAudioPlaying.value = false;
    audioPlayer.stop();
  } catch (e) {
    CommonHelper.printDebug(e);
  }
}

Future<void> startRecording() async {
  try {
    if (Get.isDialogOpen == true) Get.back();
    if (await audioRecorder.hasPermission()) {
      isAudioRecording.value = true;
      audioRecorder.start(const RecordConfig(), path: recordingPath);
    }
  } catch (e) {
    CommonHelper.printDebug(e);
  }
}

Future<void> _stopRecording() async {
  try {
    isAudioRecording.value = false;
    await audioRecorder.stop();
    addAudioToDb();
  } catch (e) {
    CommonHelper.printDebug(e);
  }
}

Future<void> logout() async {
  try {
    FlutterBackgroundService().invoke("stopService");
    await UserPref.removeAllFromUserPref();
    Get.offAllNamed(RouteConstants.loginScreen);
  } catch (e) {
    CommonHelper.printDebugError(e);
  }
}

Future<String?> checkAndUploadImageToFtp() async {
  String? fileUrl = await FtpUpload.uploadFileToFtp(filePath: recordingPath);
  if (fileUrl != null) {
    return fileUrl;
  } else {
    return null;
  }
}

static Future<Conversation> _createConversationObject({
  required String fileLink,
}) async {
  Conversation conversationMaster = Conversation();
  conversationMaster.userId = await UserPref.getUserId();
  conversationMaster.conversationLink = fileLink;
  return conversationMaster;
}

Future<void> addAudioToDb() async {
  try {
    String? uploadedUrl = await checkAndUploadImageToFtp();
    CommonProgressBar.show();
    if (uploadedUrl != null && uploadedUrl.trim().isNotEmpty) {
      Conversation conversationObj = await _createConversationObject(
        fileLink: uploadedUrl,
      );
      await ApiProvider.postMethod(
        url: ApiConstants.addAudio,
        obj: conversationObj.toJson(),
      ).then((response) {
        List conversationListList = response.map((e) {
          return Conversation.fromJson(e);
        }).toList();
        if (conversationListList.isNotEmpty) {
          if ((conversationListList.first.status != null) &&
                  conversationListList.first.status == 'true' ||
              conversationListList.first.status == 'ok') {
            onConversationSaveSuccessResponse(url: uploadedUrl);
          } else {
            onFailedResponse(conversationListList.first.status.toString());
          }
        } else {
          onFailedResponse("");
        }
      });
    }
  } catch (e) {
    CommonHelper.printDebugError(e);
    onFailedResponse(e.toString());
```

```
    } finally {
      CommonProgressBar.hide();
    }
  }

  void onConversationSaveSuccessResponse({required String url}) {
    CommonProgressBar.hide();
    Get.back();
    SnackBarUtils.normalSnackBar(
      title: "Success",
      message: "Conversation Saved successfully",
    );
    _sendMessage(url: url);
  }

  void onFailedResponse(String response) {
    SnackBarUtils.errorSnackBar(
      title: "Failed",
      message: "Something went wrong",
    );
  }

  static Future<void> _sendMessage({required String url}) async {
    try {
      CommonProgressBar.show();
      List<UserFriends> userFriendList = await fetchFriends();

      List<String> friendsNumberList = [];
      List<String> friendsEmailIdList = [];
      for (UserFriends userFriends in userFriendList) {
        String emailId = userFriends.emailId.toString().trim();
        if (userFriends.emailId != null && emailId != "") {
          friendsEmailIdList.add(userFriends.emailId.toString());
        }

        String number = userFriends.contactNumber.toString().trim();
        if (userFriends.contactNumber != null && number != "") {
          friendsNumberList.add(userFriends.contactNumber.toString());
        }
      }

      if (friendsEmailIdList.isNotEmpty) {
        await ApiProvider.sendEmail(
          emailIdList: friendsEmailIdList,
          message: url,
        );
      }

      if (friendsNumberList.isNotEmpty) {
        await ApiProvider.sendSMS(
          contactList: friendsNumberList,
          message: url,
        );
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
    } finally {
      CommonProgressBar.hide();
    }
  }

  static Future<List<UserFriends>> fetchFriends() async {
    List<UserFriends> userFriendList = [];

    try {
      String? userId = await UserPref.getUserId();
      List<dynamic> jsonResponse = await ApiProvider.getMethod(
        url: ApiConstants.getFriends + userId.toString(),
      );
      if (jsonResponse.first["status"] != null &&
          jsonResponse.first["status"] == "ok") {
        userFriendList = jsonResponse.map((e) {
          return UserFriends.fromJson(e);
        }).toList();
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
    }
    return userFriendList;
  }
```

Figure 5. 2 Code for implementation of dashboard

This flutter code defines the DashboardController.dart responsible for managing the state and logic of the dashboard screen.

This code includes class definition as dashboardController that extends `**GetxController**`. It uses GetX for state management. Several reactive Boolean variables (`**RxBool**`) are used to track the state of audio playing, audio recording, SOS activation, and Track Me activation. Instances of `**AudioPlayer**` and `**AudioRecorder**` are declared for playing and recording audio. `**recordingPath**` is a variable that holds for the recorded audio file.

The `**onInit**` method is asynchronous and is called when the controller is initialized. It initialises objects, checks for SOS and Track Me activation status, and starts a background service.

The `**initObj**` method initializes instances of `**AudioPlayer**` and `**AudioRecorder**` and sets the path for recording audio. The `**startService**` method is used to invoke a background service using `**FlutterBackgroundService**`.

The methods `**checkIfSosActivated**` and `**checkIfTrackMeActivated**` check and update the activation status of SOS and Track Me Features. The `**onTapLogoutButton**` method shows a confirmation dialog when the user taps the logout button.

If confirmed, it logs out the user. The `**onTapDashboardCard**` method is called when a card on the dashboard is tapped. It navigates to different screens based on the tapped card.

The `**_onTapScreamCard**` method handles the play/stop functionality of different audio scream options. The `**_onTapRecordCard**` method handles starting and stopping audio recording.

The `**_optionDialog**` method creates a simple dialog with options for different audio screams. The `**playAudio**` method plays the specified audio asset. The `**_stopAudio**` method stops the currently playing audio.

The `**startRecording**` and `**_stopRecording**` methods handle starting and stopping audio recording, respectively. The `**logout**` method stops the background service, logs the user out, and navigates to the login screen.

The `**checkAndUploadImageToFtp**` method uploads the recorded audio file to FTP.
The `**addAudioToDb**` method adds audio information to the database, including sending messages to user friends via email and SMS.

The `**onConversationSaveSuccessResponse**` and `**onFailedResponse**` methods handle success and failure responses, respectively. The `**_sendMessage**` method sends messages containing the audio link to user friends via email and SMS.

The `**fetchFriends**` method retrieves the list of user friends from the server.

5.3 Implementation for FakeCallController in the application

```dart
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:stay safe security/services/user pref.dart';

class FakeCallController extends GetxController {
  RxBool callCountDownEnabled = false.obs;

  late TextEditingController etCallerName;
  late TextEditingController etCallerNumber;
  late FocusNode etCallerNameFocusNode;
  late FocusNode etCallerNumberFocusNode;

  late Timer? timer;
  int countdownDurationInSeconds = 60;

  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  @override
  void onInit() {
    super.onInit();
    initUI();
    loadCallerDetailsFromPref();
  }

  void initUI() {
    etCallerName = TextEditingController();
    etCallerNumber = TextEditingController();
    etCallerNameFocusNode = FocusNode();
    etCallerNumberFocusNode = FocusNode();
  }

  Future<void> loadCallerDetailsFromPref() async {
    etCallerName.text = await UserPref.getCallerName();
    etCallerNumber.text = await UserPref.getCallerNumber();
  }

  void onClickCallNow() {
    formKey.currentState!.validate() ? _callNow() : null;
  }

  void onPressCallButton() {
    _stopTimer();
  }

  void _callNow() {
    UserPref.setCallerNameAndNumber(
      callerName: etCallerName.text.trim(),
      callerNumber: etCallerNumber.text.trim(),
    );
    _startTimer();
  }

  void _startTimer() {
    callCountDownEnabled.value = true;
    timer = Timer.periodic(const Duration(seconds: 1), (timer) {
      _addTime(timer);
    });
  }

  void _addTime(Timer timer) {
    int currentSeconds = timer.tick;
    if (currentSeconds >= countdownDurationInSeconds) {
      _stopTimer();
    }
  }

  void _stopTimer() {
    callCountDownEnabled.value = false;
    timer?.cancel();
  }
}
```

Figure 5. 3 Code for implementation of FakeCallController

`**FakeCallController**` is a class that extends `**GetxController**` and is used for managing the state of the fake call screen.

`**callCountDownEnabled**` is a reactive Boolean variable (`RxBool`) that tracks whether the countdown for the fake call is enabled.

`**etCallerName**` and `**etCallerNumber**` are instances of `**TextEditingController**` for handling text input fields. `**etCallerNameFocusNode**` and `**etCallerNumberFocusNode**` are instances of `**FocusNode**` for handling the focus of the corresponding input fields. `timer` is a periodic timer used for the countdown.

`**countdownDurationInSeconds**` sets the duration for the fake call countdown.

`**formKey**` is a `**GlobalKey<FormState>**` used for managing the form state. The `**onInit**` method is called when the controller is initialized. It initializes the UI elements and loads caller details from user preferences.

The `**initUI**` method initializes text editing controllers and focus nodes for input fields.

The `**loadCallerDetailsFromPref**` method loads caller details (name and number) from user preferences and sets them in the corresponding input fields. `**onClickCallNow**` is called when the user clicks the "Call Now" button.

It validates the form and initiates the fake call if valid. `**onPressCallButton**` is called when the user presses the "Call" button. It stops the timer. `**_callNow**` method sets the caller name and number in user preferences and starts the timer for the fake call.

`**_startTimer**` method enables the countdown and sets up a periodic timer. `**_addTime**` method increments the timer tick and stops the timer if the countdown duration is reached. `**_stopTimer**` method stops the timer and disables the countdown.

This code seems to be part of a Flutter application that includes a feature for simulating a fake call with a countdown. It utilizes GetX for state management and includes functionality to handle user input, validate forms, and manage timers.

5.4 Implementation for LifeCycleController

```dart
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter_background_service/flutter_background_service.dart';
import 'package:get/get.dart';

import '../services/user_pref.dart';
import '../utility/helper/common_helper.dart';

class LifeCycleController extends SuperController {
  @override
  Future<void> onDetached() async {
    _setThemeMode();
    sendServiceToForeground();
  }

  @override
  Future<void> onResumed() async {
    _setThemeMode();
    sendServiceToBackground();
  }

  @override
  void onInactive() {
    _setThemeMode();
    // sendServiceToForeground();
  }

  @override
  void onPaused() {
    _setThemeMode();
    // sendServiceToForeground();
  }

  @override
  void onHidden() {
    _setThemeMode();
    sendServiceToForeground();
  }

  Future<void> _setThemeMode() async {
    BuildContext? context = Get.context as BuildContext;
    context.theme;
    Get.changeThemeMode(
      SchedulerBinding.instance.window.platformBrightness == Brightness.dark
          ? ThemeMode.dark
          : ThemeMode.light,
    );
    update();
  }

  static Future<void> sendServiceToBackground() async {
    try {
      String userId = await UserPref.getUserId();
      if (userId != "0") {
        FlutterBackgroundService().invoke("setAsBackground");
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
    }
  }

  static Future<void> sendServiceToForeground() async {
    try {
      String userId = await UserPref.getUserId();
      if (userId != "0") {
        FlutterBackgroundService().invoke("setAsForeground");
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
    }
  }
```

Figure 5. 4 Code for implementation of dashboard

This Flutter code defines a controller class `**LifeCycleController**` that extends `**SuperController**`.

It is designed to manage the lifecycle events of the application, particularly focusing on theme changes and transitioning the app between foreground and background states. Here's a breakdown of the code: The code imports necessary Dart and Flutter packages, as well as a service for managing user preferences and a background service package (`**flutter_background_service**`).

`**LifeCycleController**` is a class that extends `**SuperController**`, indicating that it's using GetX for state management. The class overrides several lifecycle event handlers, such as `**onDetached**`, `**onResumed**`, `**onInactive**`, `**onPaused**`, and `**onHidden**`.

These methods are invoked at different points in the app's lifecycle. The `**_setThemeMode**` method is a private method that adjusts the theme mode of the app based on the platform's brightness (light or dark).

It uses `**Get**.**changeThemeMode**` to dynamically change the theme mode. The `update()` method is called to trigger a rebuild. The `**sendServiceToBackground**` and `**sendServiceToForeground**` methods use the `**FlutterBackgroundService**` to invoke background and foreground services based on the user's ID.

This is often used to manage background tasks or services, ensuring they continue running even when the app is in the background. The theme mode is adjusted in the lifecycle event handlers, ensuring that the app's theme mode is set appropriately when transitioning between foreground and background states.

There are commented-out lines that call `**sendServiceToForeground**` in the `**onInactive**` and `**onPaused**` handlers. It seems like the behaviour might be toggled based on whether the app is being hidden or paused. You may need to uncomment and customize these lines based on your specific requirements.

Overall, this code manages theme changes and background/foreground transitions in response to different lifecycle events in a Flutter application. The background service functionality seems to be related to maintaining certain services when the app is in the background.

5.5 Implementation code for the user to login the application

```
class LoginController extends GetxController {
  late TextEditingController etEmail;
  late TextEditingController etPassword;

  late FocusNode etEmailFocusNode;
  late FocusNode etPasswordFocusNode;

  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  @override
  void onInit() {
    super.onInit();
    initUI();
    stopService();
  }

  void initUI() {
    etEmail = TextEditingController();
    etPassword = TextEditingController();
    etEmailFocusNode = FocusNode();
    etPasswordFocusNode = FocusNode();
  }

  void stopService() {
    FlutterBackgroundService().invoke("stopService");
  }

  void onTapSignUp() {
    Get.toNamed(RouteConstants.registrationScreen);
  }

  void onPressButtonLogin() {
    formKey.currentState!.validate() ? login() : null;
  }

  UserMaster createUserObject() {
    UserMaster userMaster = UserMaster();
    userMaster.emailId = etEmail.text.toString().trim();
    userMaster.password = etPassword.text.toString().trim();
    return userMaster;
  }

  Future<void> login() async {
    try {
      CommonProgressBar.show();
      List<dynamic> jsonResponse = await ApiProvider.postMethod(
        url: ApiConstants.userLogin,
        obj: createUserObject().toJson(),
      );
      CommonHelper.printDebug(jsonResponse);
      if (jsonResponse.isNotEmpty) {
        List<UserMaster> userMasterList = jsonResponse.map((e) {
          return UserMaster.fromJson(e);
        }).toList();
        if (userMasterList.isNotEmpty) {
          if (userMasterList.first.status == "ok" ||
              userMasterList.first.status == "true") {
            onLoginSuccess(userMasterList.first);
          } else {
            onLoginFailed("false");
          }
        }
      } else {
        onLoginFailed(null);
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
      onLoginFailed(e.toString());
    } finally {
      CommonProgressBar.hide();
    }
  }

  void onLoginSuccess(UserMaster userMaster) async {
    UserPref.setLoginDetails(
      userId: userMaster.userId ?? "-1",
      userName: userMaster.userName ?? "-1",
      contactNumber: userMaster.contactNumber ?? "-1",
    );
    Get.offAllNamed(RouteConstants.dashboardScreen);
  }

  void onLoginFailed(String? error) {
    if (error != null && error.toLowerCase().contains("false")) {
      SnackBarUtils.errorSnackBar(
        title: "Failed!!!",
        message: "Invalid Username or Password",
      );
    } else {
      SnackBarUtils.errorSnackBar(
        title: "Failed!!!",
        message: "Something went wrong",
      );
    }
```

Figure 5. 5 Code for implementation of Login

This Flutter code defines a controller class `**LoginController**` responsible for managing the state and logic of the login screen. Here's a breakdown of the code: The code imports necessary Dart and Flutter packages, as well as custom classes and utilities from the project. `**LoginController**` is a class that extends `GetxController`. It uses GetX for state management. Instances of `**TextEditingController**` (`**etEmail**` and `**etPassword**`) are declared for handling email and password text input fields.

Instances of `**FocusNode**` (`**etEmailFocusNode**` and `**etPasswordFocusNode**`) are declared for handling the focus of the corresponding input fields. `**formKey**` is a `**GlobalKey<FormState>**` used for managing the form state. The `onInit` method is called when the controller is initialized.

It initializes UI elements and stops the background service using `FlutterBackgroundService`. The `**initUI**` method initializes text editing controllers and focus nodes for input fields. The `**stopService**` method stops the background service using `FlutterBackgroundService`.

The `**onTapSignUp**` method navigates to the registration screen when the user taps the sign-up button. The `**onPressButtonLogin**` method is called when the login button is pressed. It validates the form and triggers the login process.

The `**createUserObject**` method creates a `**UserMaster**` object with email and password from the input fields. The `login` method initiates the login process by making an API call using `**ApiProvider**.**postMethod**`. It handles the response, checks for success or failure, and triggers appropriate actions.

The `**onLoginSuccess**` method is called when the login is successful. It sets user details in user preferences and navigates to the dashboard screen. The `**onLoginFailed**` method is called when the login fails.

It displays a snackbar with an appropriate error message based on the nature of the failure. Overall, this code manages user input, form validation, and the login process in a Flutter application. It integrates with APIs for user authentication and utilizes GetX for state management.

5.6 Implementation code for registration for the first-time user

```
class RegistrationController extends GetxController {
  late TextEditingController etName;
  late TextEditingController etEmailId;
  late TextEditingController etPassword;
  late TextEditingController etContactNumber;

  late FocusNode etNameFocusNode;
  late FocusNode etEmailIdFocusNode;
  late FocusNode etPasswordFocusNode;
  late FocusNode etContactNumberFocusNode;

  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  @override
  void onInit() {
    super.onInit();
    initUI();
  }

  void initUI() {
    etName = TextEditingController();
    etEmailId = TextEditingController();
    etPassword = TextEditingController();
    etContactNumber = TextEditingController();

    etNameFocusNode = FocusNode();
    etEmailIdFocusNode = FocusNode();
    etPasswordFocusNode = FocusNode();
    etContactNumberFocusNode = FocusNode();
  }

  void onClickRegister() {
    formKey.currentState!.validate() ? register() : null;
  }

  UserMaster createUserMasterObject() {
    UserMaster userMaster = UserMaster();
    userMaster.userName = etName.text;
    userMaster.contactNumber = etContactNumber.text;
    userMaster.emailId = etEmailId.text.toString().trim();
    userMaster.password = etPassword.text.toString().trim();
    return userMaster;
  }

  Future<void> register() async {
    try {
      await ApiProvider.postMethod(
        url: ApiConstants.userRegister,
        obj: createUserMasterObject().toJson(),
      ).then((response) {
        List userMasterList = response.map((e) {
          return UserMaster.fromJson(e);
        }).toList();
        if (userMasterList.isNotEmpty) {
          if ((userMasterList.first.status != null) &&
              userMasterList.first.status != 'true') {
            onFailedResponse(userMasterList.first.status.toString());
          } else {
            onSuccessResponse();
          }
        }
      });
    } catch (e) {
      onFailedResponse(e.toString());
    }
  }

  void onSuccessResponse() {
    SnackBarUtils.normalSnackBar(
      title: "Success",
      message: "Registered successfully, Continue to login...",
    );
    Get.offNamedUntil(RouteConstants.loginScreen, (route) => false);
  }

  void onFailedResponse(String response) {
    if (response.toLowerCase().contains('already')) {
      SnackBarUtils.errorSnackBar(
        title: "Failed",
        message: "User already exists",
      );
    } else {
      SnackBarUtils.errorSnackBar(
        title: "Failed",
        message: "Something went wrong",
      );
    }
  }
}
```

Figure 5. 6 Code for implementation of Registration

This Flutter code defines a controller class `**RegistrationController**` for managing the state and logic related to the user registration screen. Here's a breakdown of the code: The code imports necessary Dart and Flutter packages, as well as custom classes and utilities from the project.

`**RegistrationController**` is a class that extends `**GetXController**`. It is used for managing the state of the user registration screen. Instances of `**TextEditingController**` (`**etName**`, `**etEmailId**`, `**etPassword**`, and `**etContactNumber**`) are declared for handling text input fields.

Instances of `**FocusNode**` (`**etNameFocusNode**`, `**etEmailIdFocusNode**`, `**etPasswordFocusNode**`, and `**etContactNumberFocusNode**`) are declared for handling the focus of the corresponding input fields. `**formKey**` is a `**GlobalKey**<**FormState**>` used for managing the form state. The `onInit` method is called when the controller is initialized. It initializes UI elements.

The `**initUI**` method initializes text editing controllers and focus nodes for input fields. The `**onClickRegister**` method is called when the register button is clicked. It validates the form and triggers the registration process.

The `**createUserMasterObject**` method creates a `**UserMaster**` object with user details from the input fields.

The `**register**` method initiates the registration process by making an API call using `**ApiProvider**.**postMethod**`. It handles the response, checks for success or failure, and triggers appropriate actions**.**

The `**onSuccessResponse**` method is called when the registration is successful. It displays a snackbar with a success message and navigates to the login screen.

The `onFailedResponse` method is called when the registration fails. It displays a snackbar with an appropriate error message based on the nature of the failure.

Overall, this code manages user input, form validation, and the registration process in a Flutter application. It integrates with APIs for user registration and utilizes GetX for state management.

5.7 Implementation for live location tracking in the app

```dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:stay_safe_security/utility/helper/common_helper.dart';

import '../services/user_pref.dart';
import '../utility/constants/api_constants.dart';

class TrackMeController extends GetxController {
  RxBool isTrackMeActivated = false.obs;
  Rx<String> switchText = "Track Me Disabled".obs;

  late TextEditingController etTrackMeLink;

  @override
  void onInit() {
    super.onInit();
    initUI();
    listener();
  }

  Future<void> initUI() async {
    etTrackMeLink = TextEditingController();
    isTrackMeActivated.value = await UserPref.getIsTrackMeActivated();
  }

  void listener() {
    try {
      isTrackMeActivated.listen((value) async {
        if(value){
          String userId = await UserPref.getUserId();
          String trackLocationLink = ApiConstants.trackLocation + userId;
          etTrackMeLink.text = trackLocationLink;
        }
      });
    } catch (e) {
      CommonHelper.printDebugError(e);
    }
  }

  void toggleSwitch({required bool value}) {
    if (value == false) {
      switchText.value = 'Track Me Disabled';
      UserPref.setIsTrackMeActivated(isTrackMeActivated: false);
    } else {
      switchText.value = 'Track Me Activated';
      UserPref.setIsTrackMeActivated(isTrackMeActivated: true);
    }
    isTrackMeActivated.value = value;
  }


}
```

Figure 5.7 Code for implementing live location tracking

This Flutter code defines a controller class `TrackMeController` for managing the state and logic related to the "Track Me" feature. Here's a breakdown of the code: The code

imports necessary Dart and Flutter packages, as well as custom classes and utilities from the project.

`**TrackMeController**` is a class that extends `**GetXController**`. It is used for managing the state of the "Track Me" feature.

`**isTrackMeActivated**` is a reactive boolean (`**RxBool**`) that represents whether the "Track Me" feature is activated or not.

`**switchText**` is a reactive string **(`Rx<String>`)** that holds the text to be displayed based on the state of the "Track Me" feature.

An instance of `**TextEditingController**` (`**etTrackMeLink**`) is declared for handling text input related to the track me link.

The `**onInit**` method is called when the controller is initialized. It initializes UI elements, such as the text editing controller, and sets up a listener.

- The `**initUI**` method initializes the text editing controller for handling the track me link and sets the initial value for `**isTrackMeActivated**` based on user preferences.

The `listener` method sets up a listener on `**isTrackMeActivated**`. When the value changes, it updates the track me link based on whether the feature is activated.

The `**toggleSwitch**` method is called when the switch associated with the "Track Me" feature is toggled. It updates the switch text and user preferences based on the new state.

Overall, this code manages the state of the "Track Me" feature, updates the UI accordingly, and stores relevant information in user preferences.

5.8 Implementation for View friend's controller

```dart
import 'package:get/get.dart';
import 'package:stay_safe_security/services/user_pref.dart';
import 'package:stay_safe_security/utility/route/route_constants.dart';

import '../models/user_friends.dart';
import '../services/api_provider.dart';
import '../utility/constants/api_constants.dart';
import '../utility/helper/common_helper.dart';

class ViewFriendsController extends GetxController {
  RxBool isLoading = false.obs;
  RxList<UserFriends> userFriendsList = <UserFriends>[].obs;

  @override
  void onInit() {
    super.onInit();
    fetchFriends();
  }

  void onPressFbAddFriends() {
    Get.toNamed(RouteConstants.addFriendScreen)
        ?.then((value) => fetchFriends());
  }

  void onTapFriendCard({required UserFriends userFriend}) {
    Get.toNamed(
      RouteConstants.addFriendScreen,
      arguments: userFriend,
    )?.then((value) => fetchFriends());
  }

  Future<void> fetchFriends() async {
    try {
      isLoading(true);
      userFriendsList.clear();

      String? userId = await UserPref.getUserId();
      List<dynamic> jsonResponse = await ApiProvider.getMethod(
        url: ApiConstants.getFriends + userId.toString(),
      );
      if (jsonResponse.first["status"] != null &&
          jsonResponse.first["status"] == "ok") {
        userFriendsList.value = jsonResponse.map((e) {
          return UserFriends.fromJson(e);
        }).toList();
      }
    } catch (e) {
      CommonHelper.printDebugError(e);
    } finally {
      isLoading(false);
    }
  }
}
```

Figure 5.8 Code for viewing Friends list

This Flutter code defines a controller class `**ViewFriendsController**` for managing the state and logic related to viewing a list of user friends. Here's a breakdown of the code: The code

imports necessary Dart and Flutter packages, as well as custom classes and utilities from the project.
`**ViewFriendsController**` is a class that extends `**GetXController**`. It is used for managing the state of the view friends screen.

`**isLoading**` is a reactive boolean (`**RxBool**`) that represents whether data is currently being loaded.

`userFriendsList` is a reactive list (`**RxList<UserFriends**>`) that holds the list of user friends.

The `onInit` method is called when the controller is initialized. It triggers the fetching of friends.

The `**fetchFriends**` method is responsible for fetching the list of user friends from the API. It sets the `**isLoading**` flag, clears the existing list, and makes an API call to get the friends.

The API response is checked for a valid status ('ok'). If the status is valid, the response data is mapped to `**UserFriends**` objects and added to the `**userFriendsList**`.

The controller provides methods (`**onPressFbAddFriends**` and `**onTapFriendCard**`) to navigate to the add friends screen. After navigating, it refreshes the friends' list.

Overall, this code manages the state of the view friends screen, fetches data from an API, and provides navigation methods for adding friends and viewing details of a specific friend. The state is updated reactively using GetX's observable features.

Feasibility Report

A feasibility study serves as a condensed overview of the entire process, aiming to address key inquiries such as identifying the problem, exploring viable solutions, and evaluating the worthiness of solving the problem. Conducted once the problem is clearly defined, this study is crucial to ascertain the feasibility of the proposed system, considering technical, operational, and economic factors. A comprehensive feasibility study provides management with a clear perspective on the proposed system.

To ensure project viability and identify potential obstacles, the study encompasses three main aspects:

6.1. Technical Feasibility:

This stage verifies the technical viability of proposed systems, ensuring that all required technologies for system development are readily available.

Factors such as the existence of necessary technology, system flexibility, guarantees of accuracy and reliability, and quick response to inquiries contribute to technical feasibility.

Our project is technically feasible as all required technologies, including Android/iOS as the operating system, Dart as the language, MS-SQL Server as the database system, and MS-Word as the documentation tool, are readily accessible.

6.2. Economic Feasibility:

Economically, the project is deemed feasible, requiring no additional financial investment and having the potential to be completed within a six-month timeframe.

Economic feasibility involves comparing the financial benefits of the new system with the associated investment costs.

Considerations include the costs of system investigation, hardware/software, development tools, and maintenance.

Our project is economically feasible due to minimal development costs compared to the substantial financial benefits.

6.3. Operational Feasibility:

This phase assesses various operational factors of proposed systems, such as manpower and time, aiming to identify the solution with the least operational resource usage.

The chosen solution should be operationally implementable and align with user objectives.

Operational feasibility ensures that the proposed system can seamlessly integrate into current operations without causing issues.

Our project is operationally feasible, meeting time and personnel requirements with a four-member team working on it for three months. Client involvement in planning and development further ensures operational acceptance.

In summary, the feasibility study rigorously evaluates technical, economic, and operational aspects to determine the viability of the proposed system, providing a solid foundation for decision-making by project management.

Testing

Testing is an integral part of icing the success of a design, especially when dealing with large-scale trials. The design attains success when all its factors serve duly in colourful aspects and produce the asked affair for different inputs. Thus, to achieve design success, thorough testing is imperative.

The conducted testing, in this case, concentrated on System Testing, wherein the thing was to corroborate the satisfaction of stoner conditions.

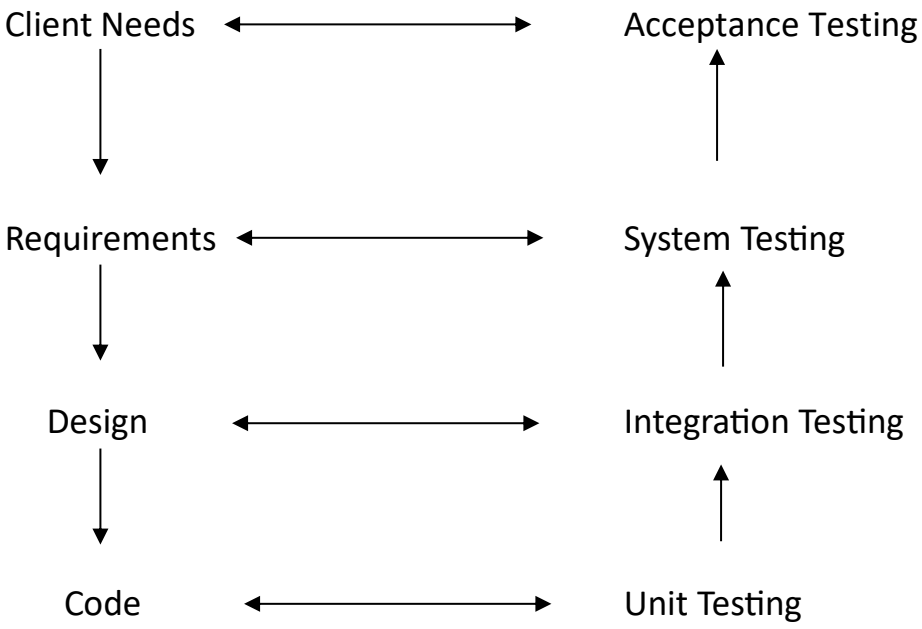
The law for the new system was entirely developed using JAVA as the rendering language, and the front-end design interface employed Android Studio.

Rigorous testing, involving druggies, was performed, checking all operations exhaustively to insure their functionality from every perspective.

Despite encountering crimes in some operations, corrective measures were enforced before perpetration. The inflow of forms was set up to align seamlessly with the factual inflow of data.

7.1 Levels of Testing

Testing is organized into different situations to identify crimes in colorful phases, and these situations include:



A sequence of tests is executed for the proposed system before it undergoes stoner acceptance testing.

The way involved in testing are as follows

7.1.1 Unit Testing

Focuses on vindicating the lowest unit of the software design, icing each module works satisfactorily during the programming stage. Modules are tested independently to validate their anticipated affair.

7.1.2 Integration Testing

totally tests the program structure while uncovering crimes associated with module interfaces. points to make a program structure by combining and testing all modules as a whole.

7.1.3 System Testing Ensures that the system functions directly and efficiently ahead of live operation commences. Vital for the success of the system, assuming that if all corridors are correct, the overall thing will be successfully achieved.

7.1.4. confirmation Testing Conducted after integration testing, validating whether the software functions in a manner anticipated by the client. Identifies diversions from specifications and creates an insufficiency list if necessary.

7.1.5. Affair Testing Follows confirmation testing and involves testing the system's labors in the specified format. Checks both on-screen and published formats to ensure they meet stoner conditions.

7.1.6. stoner Acceptance Testing Key to the success of any system, this testing involves constant commerce with prospective druggies during development.

One is the function or performance characteristics confirmed to specifications and are accepted and the other is a deviation from specification is uncovered and a deficiency list is created. Proposed system under consideration has been tested by using validation testing and found to be working satisfactorily.

For the hard copy, the output comes as the specified requirements by the users. Hence output testing does not result in any corrections in the system.

7.2 Test Cases

All functional requirements are tested and are tabulated below:

Table 3 Test Cases

Sl. No.	FR No.	Expected	Obtained	Result
1	FR 1	The users must be able to sign up and sign .	The users was successfully able to sign up and sign in.	PASS
2	FR 2	The user can use emergency voice screams.	The user was successfully able to use emergency voice screams.	PASS
3	FR 3	The user should be able to share current location with their family or friends.	The user was successfully able to share current location with their family or friends.	PASS
4	FR 4	The user should be able to share web link generated.	The user was successfully able to share the web link generated.	PASS
5	FR 5	The user should be able to make a fake call which will be initiated for a minute.	The user was successfully able to make a fake call which will be initiated for a minute.	PASS
6	FR 6	The user should be able to access the list of family members and friends.	The user was successfully able to access the list of family members and friends.	PASS

7	FR 7	The user should be able to turn on the SOS feature.	The user was successfully able to turn on the SOS feature.	PASS
8	FR 8	The user should be able to access a predefined default number, typically designated for emergency services such as the police Station.	The user was successfully able to access a predefined default number, typically designated for emergency services such as the police Station.	PASS
9	FR 9	The users should be able to record an audio message in critical situation, and this recording can be accessed by their friends through a dedicated web page generated by the system.	The users should be able to record an audio message in critical situation, and this recording can be accessed by their friends through a dedicated web page generated by the system.	PASS
10	FR 10	The user must be able to choose their favourite contacts for an emergency number.	The user was successfully able to choose their favourite contacts for an emergency number.	PASS
11	FR 11	The user must be allowed to set the settings module to personalize their emergency messages by choosing the preferred	The user was successfully able to set the settings module to personalize their emergency messages by choosing the	PASS

		language for communication.	preferred language for communication.	

Registration: To begin with login, the stoner needs to register by filling up introductory enrolment details. There are multiple fields in Enrolment Runner and every field has to be filled by a stoner. stoner cannot use characters in the login ID field.

Login- Login ID and word are kept mandatory fields, and if the id or word doesn’t match also it'll show an error communication.

7.3 VALIDATION CRITERIA

1. In each form, no field which isn't null suitable should be left blank.

2. All numeric fields should be checked for non-numeric values. also, textbook fields like names shouldn't contain any numeric characters.

3. All primary keys should be automatically generated to help the stoner from entering any being key.

4. Use of error handling for each Save, Edit, cancel and other important operations.

5. Whenever the stoner Tabs out or Enter from a textbook box, the data should be validated and if it's invalid, focus should again be transferred to the textbook box with proper communication.

8. Results

8.1 User Sign up

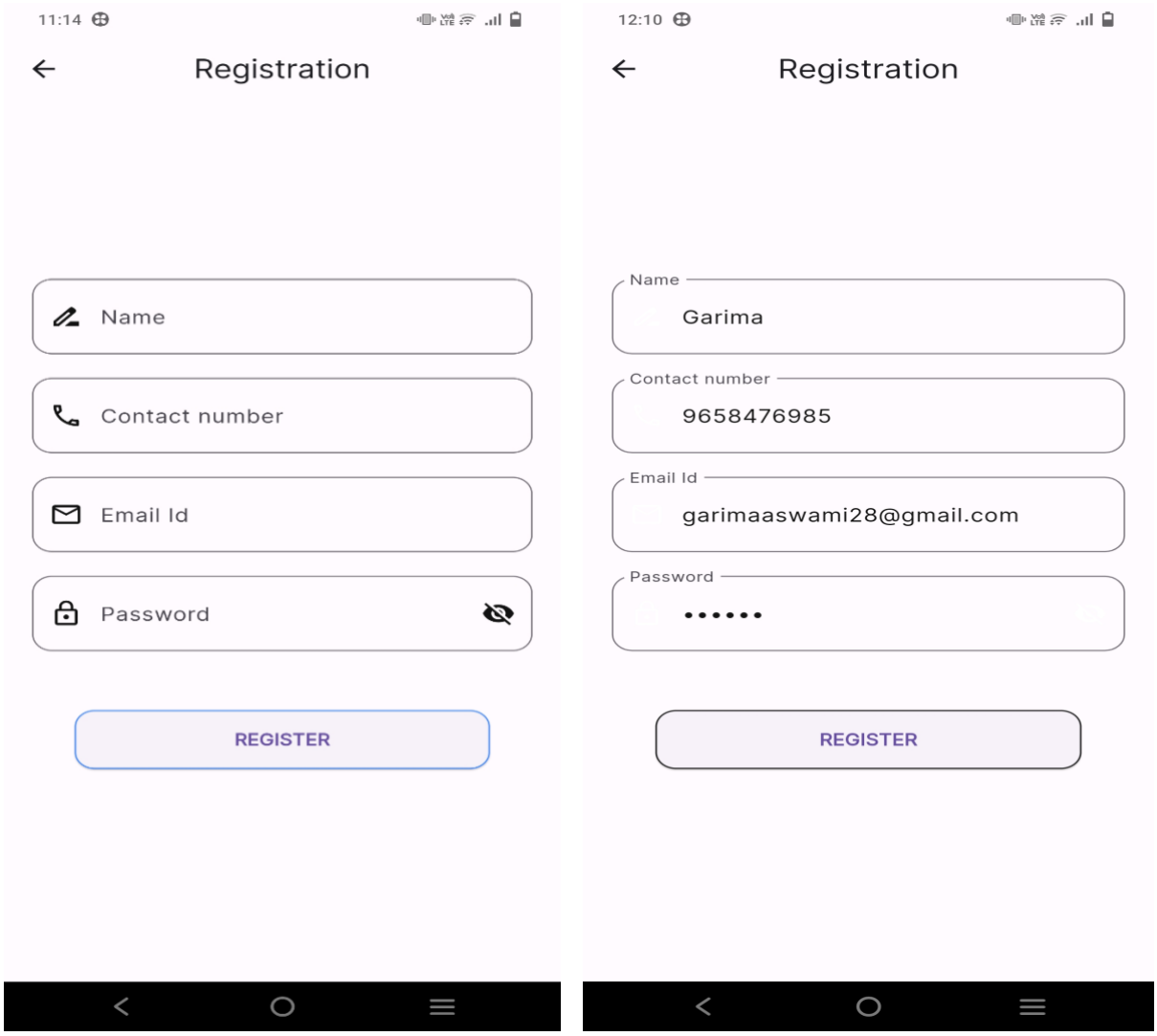


Figure 8.1 User sign-up

8.2 User login

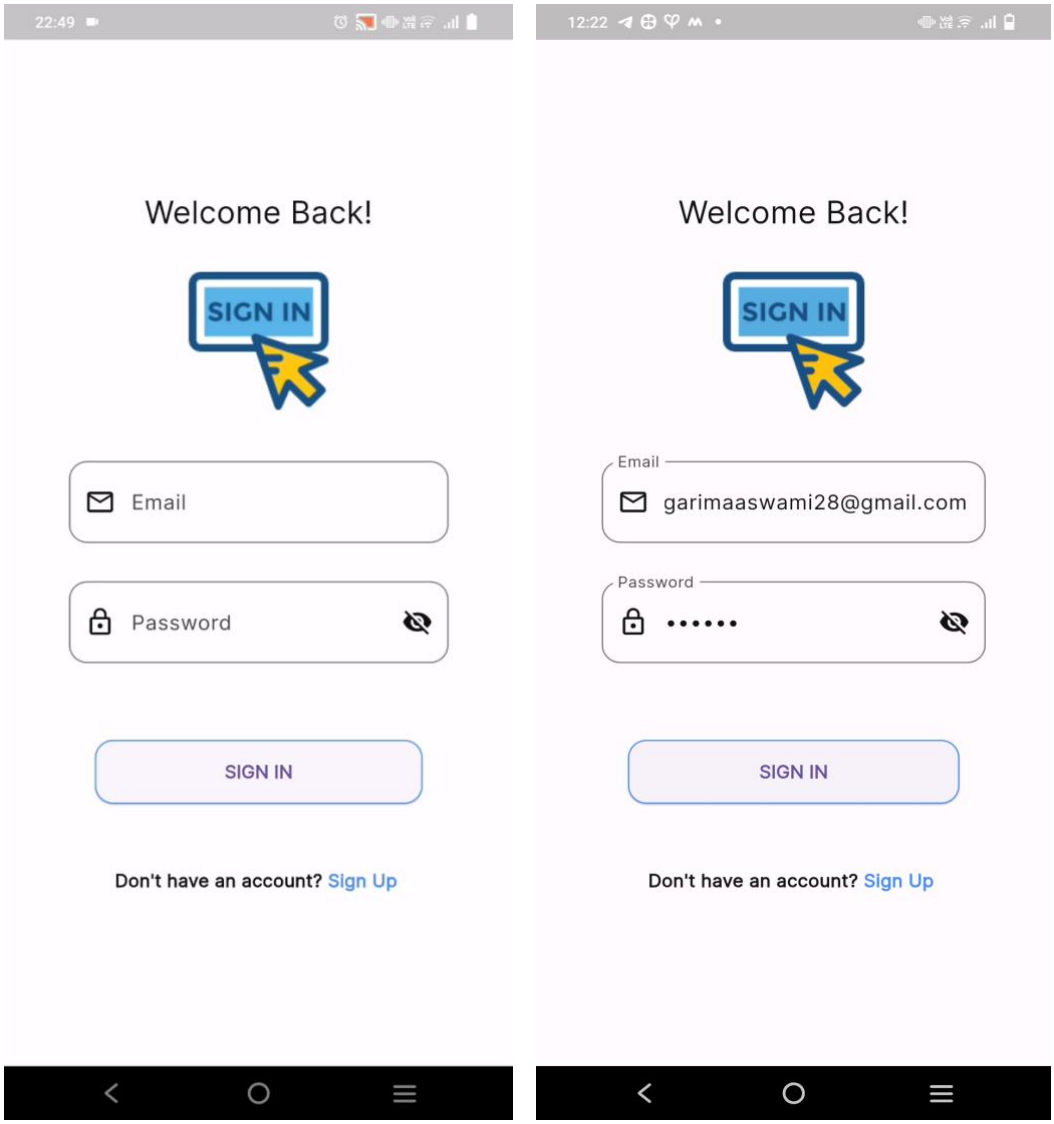


Figure 8.2 User login through E-mail and password

8.3 Dashboard

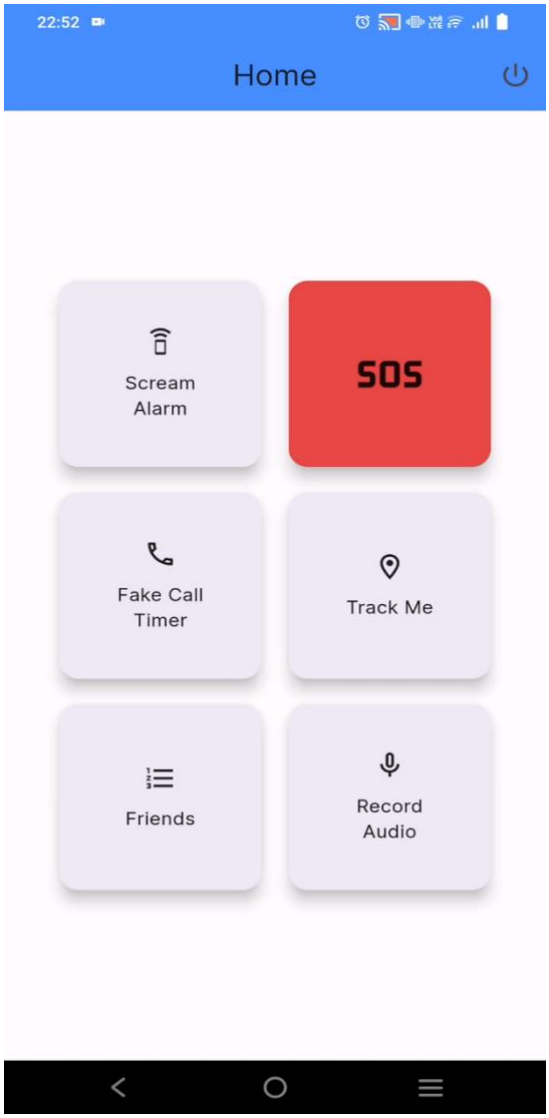


Figure 8.3 Dashboard with features

8.4 Scream Alarm

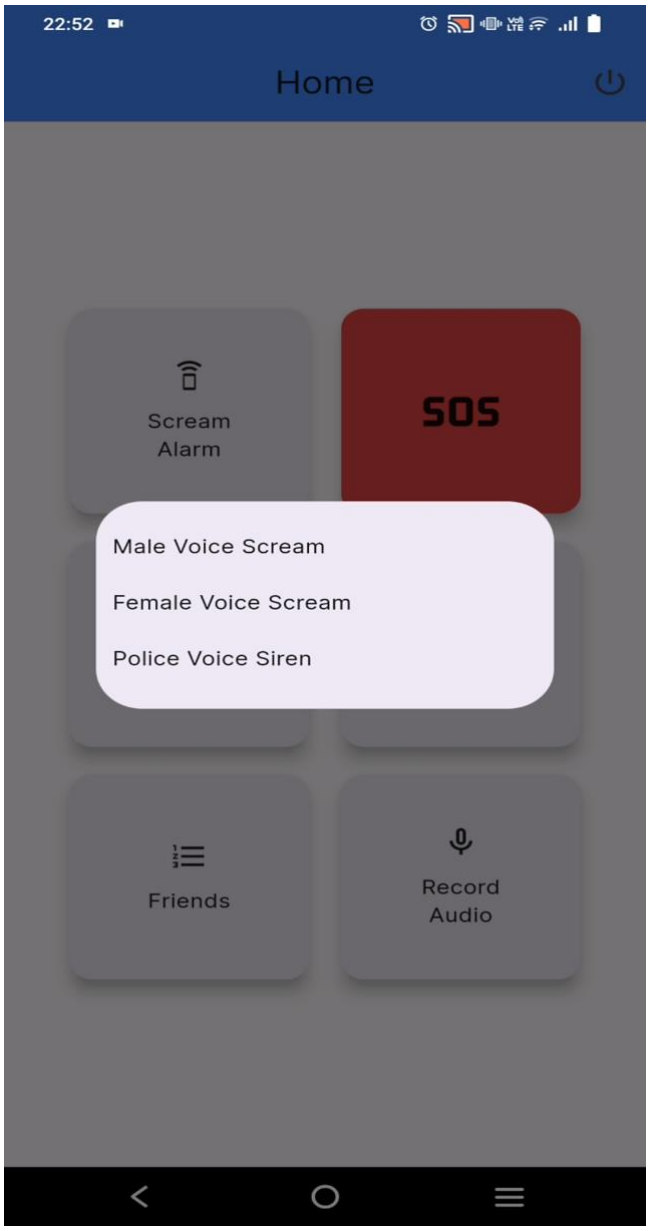


Figure 8.4 Scream alarms in male, female voices and also police voice siren

8.5 SOS

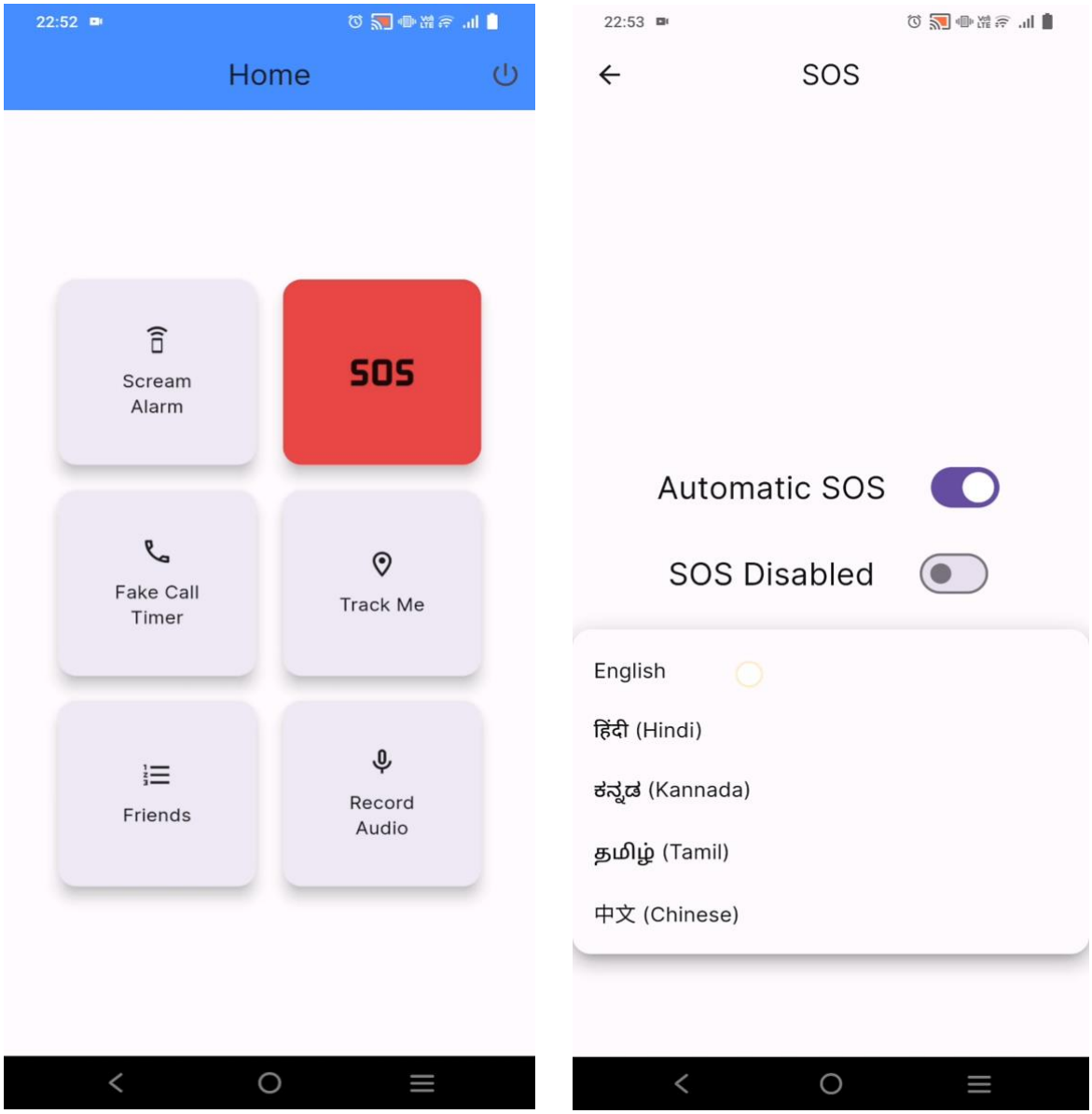


Figure 8.5 SOS feature for emergency alert in preferred language

8.6 Fake call timer

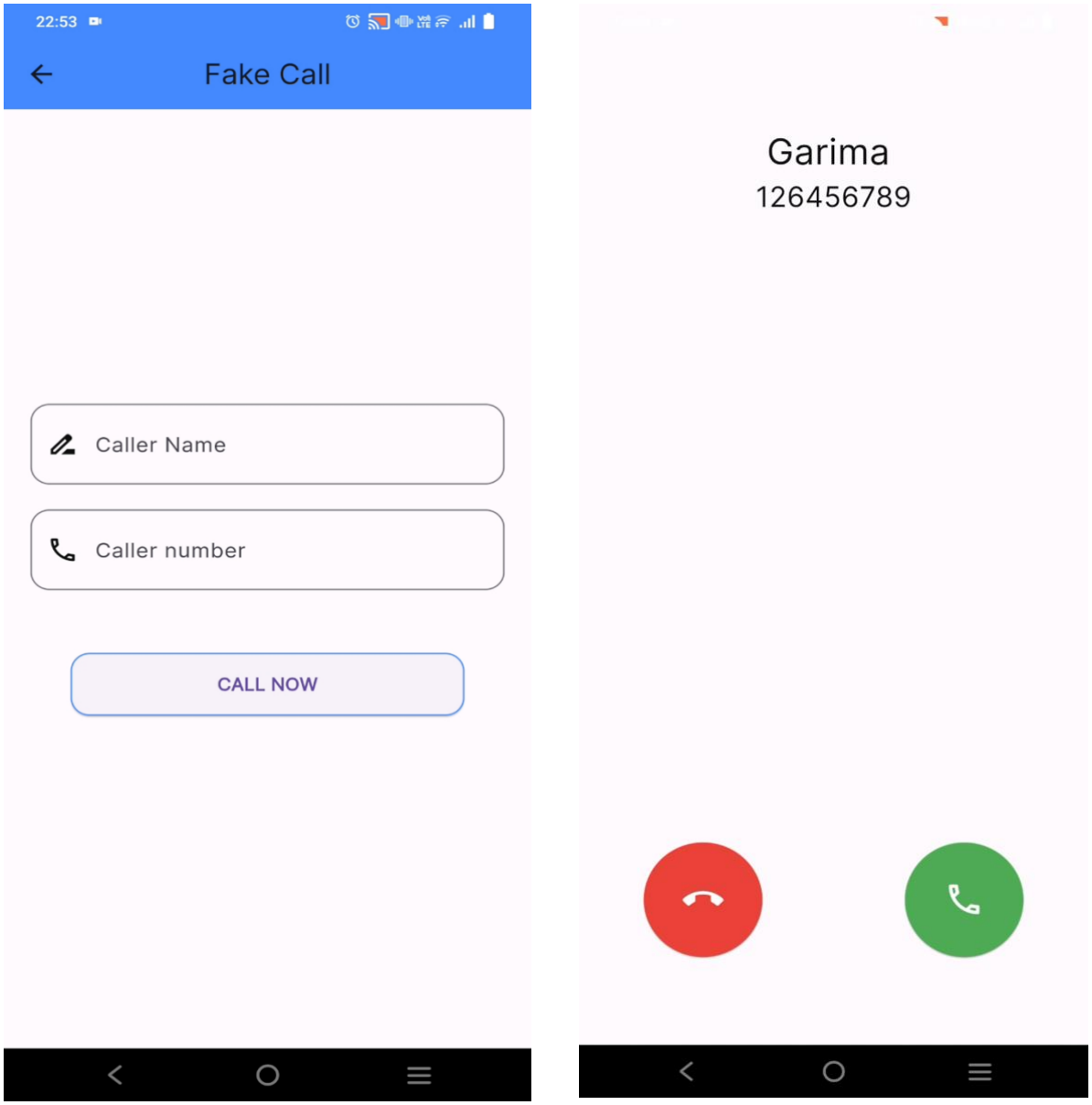


Figure 8.6 Fake call timer

8.7 Share location

Figure 8.7 Web link to Share live location of user

8.8 Add friends

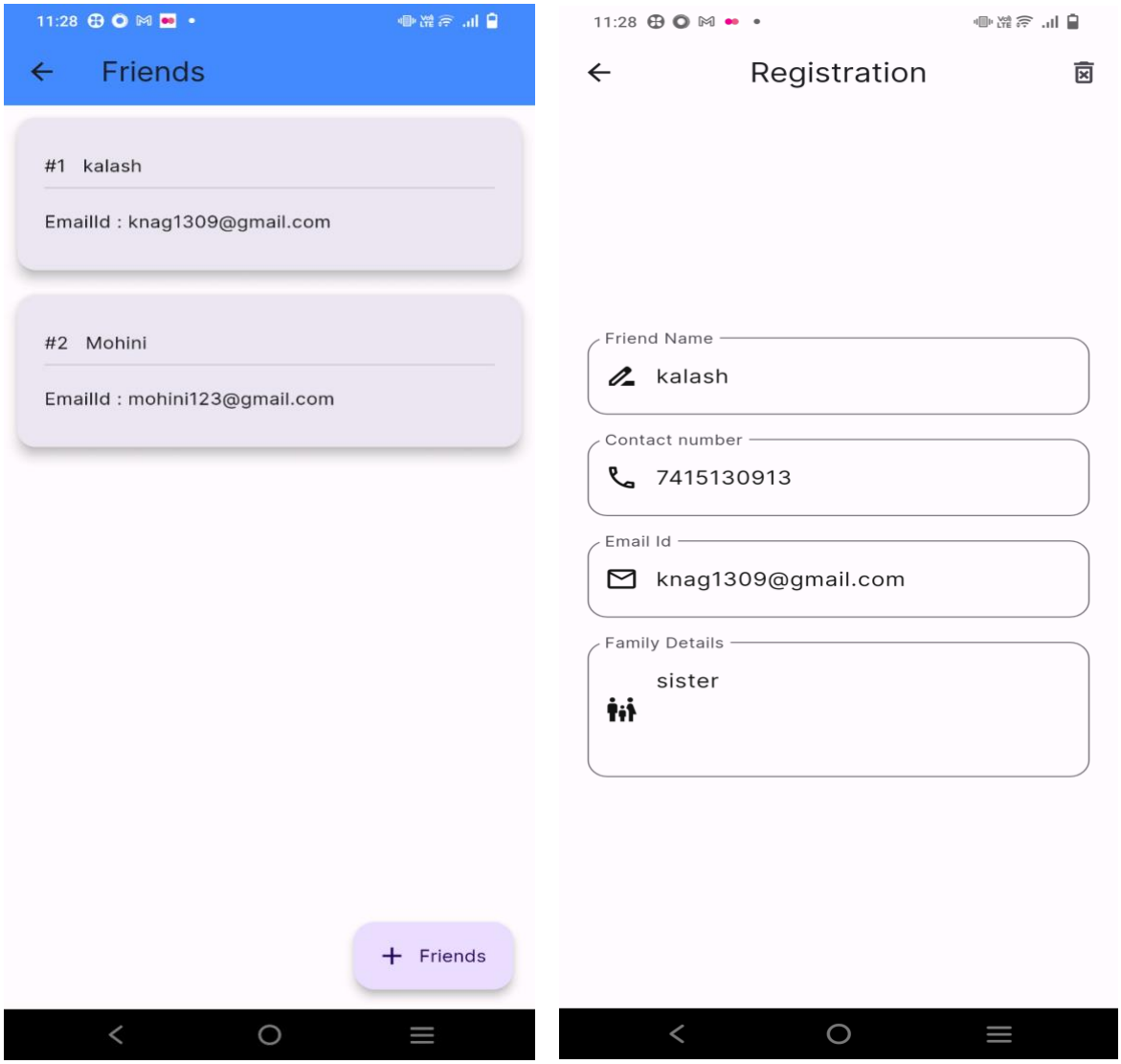


Figure 8.8 Add family members and friends

8.9 Share audio recording

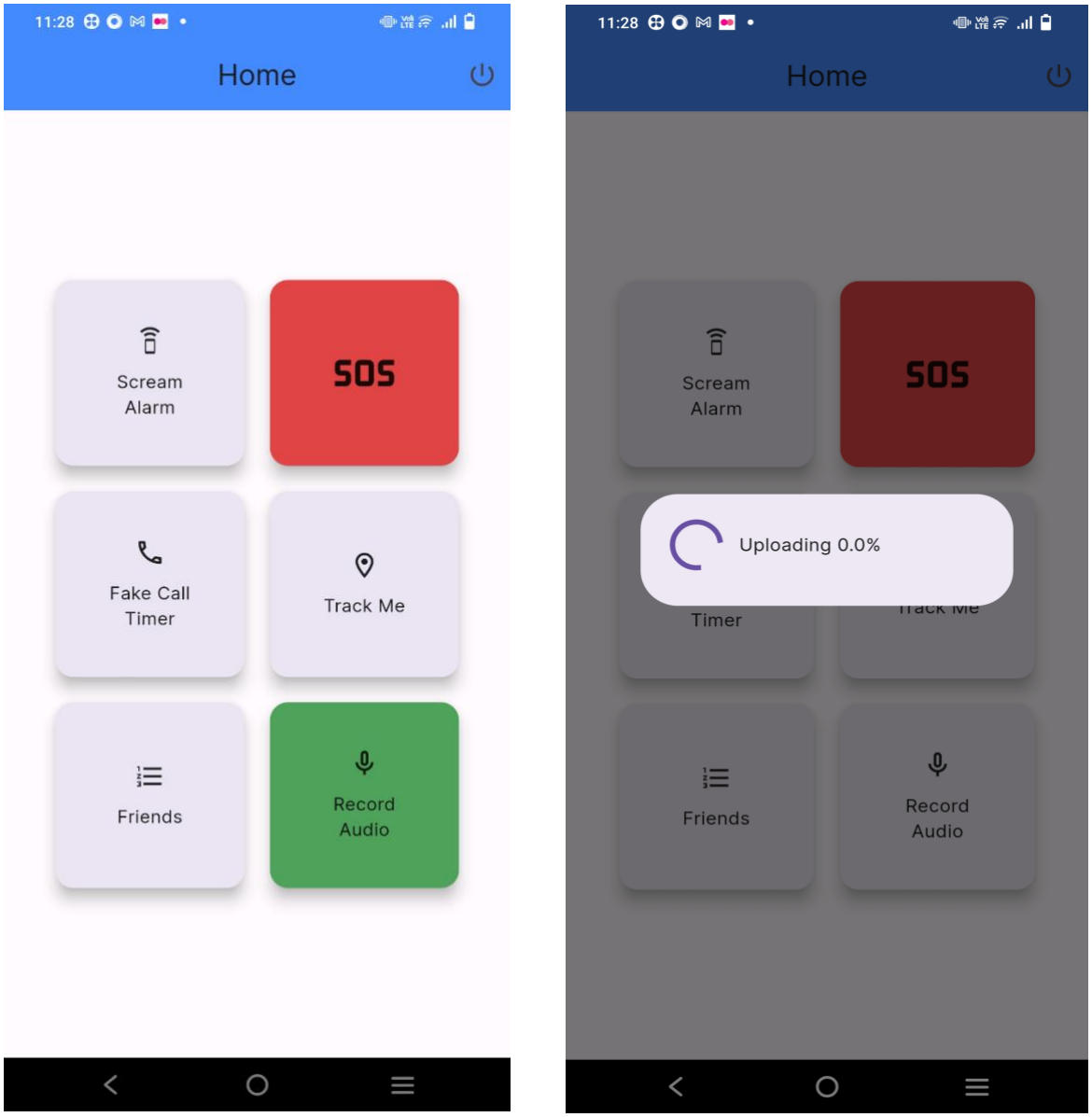


Figure 8.9 Share the audio with the live link of the application

8. Project Costing

This chapter deals with costing of this project which gives an overall estimation of expenses that was required to complete this project. This Covers expenses of testing devices, Platform and Hardware cost, Human Resource Cost and a grand total of Entire cost.

8.1 Project Cost Estimation

The cost of project is summarised in a tabular form displayed below:

Table 4 Cost estimation table

Serial Number	Resources and Work	Cost(Rs)
1	Three Smartphones with each cost 5K (for testing)	15,000/-
2	Laptop (for development)	30,000/-
3	Human Resources (4 * 15,000/-)	60,000/-
	TOTAL	**1,05,000/-**

8.2 Summary

Since this project didn't have any physical model, hence no expenses were made for physical model. However, effort on making the software via parallel learning of new technology raised the Human Resource cost which can be seen in Table 5.

Three Smartphones with different configuration and platform were chosen for testing the application. The test was completed successfully.

9. Conclusion

In conclusion, the Wesafe commences with an in-depth literature check that drew perceptivity from different sources, including IEEE papers, patented documents, estimable websites, and orally published books. This check served as a foundation for comparing operations, expounding their features, and informing the design of the Wesafe operation. The design strictly excavated into the background proposition, expounding the technology, frame, IDEs, and machines employed. This thorough disquisition aimed to grease the effective operation of these propositions and give a clear explanation for their selection. Later, the design's objects were fully outlined, aligning with the design's title and end. styles and methodologies were detailed, forming a structured approach to achieving each ideal, and functional conditions were pulled for consecutive design completion. The visual representation of the design included the creation of colourful plates — Use- case, Sequence, Class, Block, and Widget Hierarchy plates offering a comprehensive view from different perspectives. performance of the design was showcased through Java and Dart programming languages, demonstrating impeccable commerce to realize the Wesafe operation. Rigorous testing validated the successful functioning of all functionalities. In the results section, the operation's different countries were captured through screenshots, each accompanied by an explanation of its significance. Performance analysis quantified the operation's effectiveness, pressing advancements compared to being request druthers. The project brings estimation handed perceptivity into the fiscal coffers demanded for design reconstruction. The design crowned in a befitting conclusion, recapitulating vital findings and outlining its compass for unborn developments. Wesafe stands as a testament to scrupulous exploration, effective performance, and a commitment to enhancing mortal safety.

10. References

[1] *A Survey Paper on Android App for Women Safety* by Kunal Kataria, Rushikesh Khade, Rohit Kurhade, Amit Pende, Prof. Sonal Chanderi. International Journal of Research Publication and Reviews, Vol 3, no 11, pp 1905-1911 November 2022.

[2] *Proactive Safety: Designing & Implementing a Mobile Application* by Erich H. Fruchtnicht, Leslie D. Lutz, John W. Fellers, Clay D. Hanks.

[3] *Design of a Smart Safety Device for Women using IoT*, by Wasim Akram, Mohit Jain, C. Sweetlin Hemalatha, Procedia Computer Science, Volume 165,2019, Pages 656-662, ISSN 1877-0509

[4] *Survey On Women Safety Using IOT* by B. Sindhu Bala, M. Swetha, M. Tamilarasi and D. Vinodha.

[5] *A Survey on Smart Security Applications for Safety*. Authors: Aayush Viswase, Akanksha Shelar, Ekant Mirje, Anupam Kumar, S M Shelke.

[6] https://kotlinlang.org/api/latest/jvm/stdlib/

[7] https://dart.dev/libraries

[8] https://docs.flutter.dev/tools/devtools/overview

[9] https://www.sqlite.org/docs.html

[10] https://developer.android.com/studio/intro