The Last Echo

A. **Prof A.A.Patil, Professor, SKN Sinhgad Institute Of Technology & Science, Kusgaon(BK), Lonavala, India,**

B. **Mr. Vishwajit Wagh, Student, SKN Sinhgad Institute Of Technology & Science, Kusgaon(BK), Lonavala, India,**

C. **Mr. Raj Lamb, Student, SKN Sinhgad Institute Of Technology & Science, Kusgaon(BK), Lonavala, India,**

D. **Ms. Renuka Kale, Student , SKN Sinhgad Institute Of Technology & Science, Kusgaon(BK), Lonavala, India,**

E. **Mr. Prashant Take , Student, SKN Sinhgad Institute Of Technology & Science, Kusgaon(BK), Lonavala, India,**

Abstract: The Last Echo is an innovative Java-based Android application that enables users to create and store personalized messages that are sent to loved ones upon their death. By integrating Aadhaar-based authentication, the application ensures that messages are automatically dispatched once a user's Aadhaar credentials are deactivated, providing a secure and reliable posthumous communication method. The platform leverages technologies such as Spring Boot, PostgreSQL, and Android SDK to manage message creation, storage, and delivery. Prioritizing data privacy, encryption, and legal compliance, The Last Echo offers a compassionate, secure, and user friendly solution for ensuring that a user's final words are shared with dignity

Keywords: Posthumous messaging, Final message delivery, Aadhaar integration, Biometric authentication, Java Android application, Secure message storage, Message encryption, UIDAI (Unique Identification Authority of India), Privacy and data protection

I. INTRODUCTION

The Last Echo is a Java-based Android application designed to allow users to send personalized, precomposed final messages to their loved ones upon their death. The app integrates securely with Aadhaar for user authentication, ensuring that the messages are automatically delivered when the user's Aadhaar credentials are deactivated following their passing. This project focuses on providing a user friendly interface for message composition, encrypted message storage, and a reliable delivery mechanism triggered by Aadhaar deactivation. By utilizing Spring Boot for the backend, PostgreSQL for secure data storage, and Android as the frontend, the application offers a comprehensive solution for managing final communications. The app emphasizes security, data privacy, and compliance with legal standards, making sure users' last words are conveyed with care, accuracy, and dignity. By providing a compassionate platform for users to express their final words, The Last Echo addresses a significant gap in the current digital landscape. The app empowers individuals to leave behind messages of love, advice, or even instructions for their loved ones, ensuring that their final thoughts are conveyed with care and dignity. This combination of 9 technical sophistication and emotional sensitivity makes The Last Echo a unique and meaningful solution for secure posthumous
communication. A major focus of the project is the user friendly interface for composing and managing these messages. The app makes it simple for users to draft heartfelt communications and store them in an encrypted format, protecting their privacy. Encryption ensures that only the intended recipients will access these messages, adding an extra layer of security to the app. By maintaining privacy and data integrity, "The Last Echo" aligns with ethical and legal standards in handling sensitive information, giving users peace of mind. It is more than just an app; it's a thoughtful ser vice for end-of-life communication. By combining secure authentication, encrypted storage, and an intuitive user experience, it provides users with peace of mind that their final messages will reach loved ones accurately and with dignity. This application offers a unique and meaningful way for individuals to say their last goodbyes, creating a lasting impact on both the users and their families

II. LITERATURE SURVEY

Posthumous communication systems have been the subject of increasing interest as society becomes more digitally interconnected. Existing platforms like Dead Man's Switch and If I Die App allow users to create messages that are delivered after their death, usually triggered by a period of user inactivity. While innovative, these platforms lack robust verification mechanisms to confirm the user's death, which can lead to either premature message delivery or failure to deliver entirely. Similarly, social media platforms such as Facebook offer memorialization features, allowing profiles to be managed posthumously, but these are limited in scope and rely on user-reported evidence rather than verified death data. These limitations highlight the need for more reliable, secure, and usercentered systems that can effectively manage digital communication after death.

In the Indian context, Aadhaar—the Unique Identification Authority of India's biometric-based digital identity system—has emerged as a critical infrastructure for identity verification. Aadhaar is now used in various sectors, including banking, healthcare, and public services. More recently, the Indian government has introduced procedures for deactivating Aadhaar numbers upon death, based on municipal and state-level death registration services. This deactivation process serves as a verifiable and secure confirmation of death, offering a unique opportunity to use Aadhaar status as a trigger mechanism for automating sensitive processes like posthumous message delivery. The integration of Aadhaar deactivation in such systems is novel and ensures a level of authenticity and reliability that other systems lack.

From a technical standpoint, secure digital messaging relies heavily on well-established encryption standards and privacy-preserving protocols. Technologies such as end-to-end encryption (E2EE), AES-256 encryption, and secure data transfer via HTTPS are widely used to ensure the confidentiality of communications. Legal frameworks like the General Data Protection Regulation (GDPR) and India's emerging Personal Data Protection Bill (PDPB) emphasize data minimization, purpose limitation, and informed user consent. These considerations are essential when designing systems that handle emotionally sensitive and legally significant data, such as final messages sent after death. Thus, any platform offering posthumous messaging must prioritize compliance with these privacy standards. In terms of the development stack, technologies like Spring Boot, PostgreSQL, and the Android SDK provide a robust foundation for building secure and scalable applications. Spring Boot simplifies backend development and integrates seamlessly with Spring Security for authentication and authorization. PostgreSQL offers a reliable relational database management system with support for advanced encryption and access control features. The Android SDK is the standard framework for creating intuitive and performant mobile applications, with native support for integrating biometric authentication and secure local storage. These technologies together support the seamless operation of a system like The Last Echo, enabling efficient message creation, encrypted storage, and timely delivery.

Finally, the ethical and psychological dimensions of posthumous messaging cannot be overlooked. Research in the field of than technology—the intersection of death and digital technology—shows that receiving messages from deceased individuals can aid in the grieving process and provide emotional closure. However, there are ethical concerns regarding the emotional impact of such messages on recipients, as well as the importance of clear user consent and timing. Therefore, systems like The Last Echo must offer user-controlled settings for message delivery, including options for timing, recipient selection, and the ability to revoke or update messages during the user's lifetime.

III. PROPOSED SYSTEM

Hardware Requirements

1. Development Hardware Requirements : Developer Workstation (Laptop/PC): Processor: Intel Core i5 or higher, or equivalent (AMD Ryzen 5 or higher) RAM: 8 GB minimum (16 GB recommended for smoother multitasking) Storage: At least 256 GB SSD (for faster build times and application compilation) Graphics: Integrated graphics are usually sufficient for Android development, but a discrete GPU (NVIDIA or AMD) will speed up emulators. Operating System: Windows 10/11, macOS, or Linux (Ubuntu or similar).

2. Server-Side Hardware Requirements (for hosting backend) : This depends on whether you want to host the backend on your own in frastructure or use cloud-based services like AWS, GCP, or Azure. Self-hosted Server (On-premises or VPS): Processor: Intel Xeon or AMD EPYC (Quad-core or higher) RAM: 4 GB minimum (8 GB or higher recommended depending on user load) Storage: SSD storage (at least 100 GB for databases and application data) 15 Network: Reliable internet connection with good upload speed. OS: Linux (Ubuntu, CentOS) or Windows Server. For higher loads, you may need to scale the instance based on the number of users.

3. End-user Device Requirements (Android Devices) : Android Version: Android 6.0 (Marshmallow) and above (depending on SDK version you're using) RAM: 2 GB or higher(the app should be optimized to run on lowerend devices too) Storage: The app itself may not require much space, but users will need enough storage to store messages. Internet Connection: Stable internet for receiving notifications and delivering messages after deactivation.

4. Other Hardware Considerations : Aadhaar Integration: If Aadhaar authentication is done via biometric hardware (fingerprint scanner, iris scanner), additional hardware may be required for testing. Otherwise, if you're relying on OTP-based or other digital methods, this is not needed. 5. For Mobile Phones : RAM: Minimum: 2 GB RAM. Recommended: 4 GB RAM or more. Storage: Minimum: 100 MB of free space for app installation. Space will also be required for storing user-composed messages and any media files they may attach (audio, video, images). More space is advisable if users plan to store larger files. At least 1 GB of free space for the app, messages, 16 and other data.

Internet Connectivity: Stable 3G/4G or Wi-Fi connection for syncing data, receiving notifications, and verifying Aadhaar details. Security Features: Fingerprint Sensor or Biometric Authentication or NFC (Near Field Communication) Camera: If users want to attach media files (such as a video or photo) with their messages, having a phone with a decent camera (8MP or higher) can be recommended.

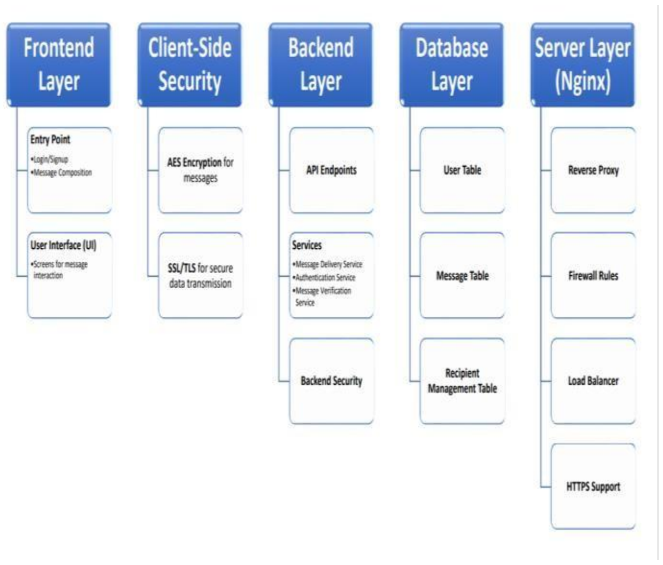
Software Requirements

1.Software Requirements for PC/Laptop (Development Server side) : Operating System: Windows: Windows 10 or higher (64-bit). MacOS: macOS 10.14 (Mojave) or higher. Linux: Ubuntu 18.04 LTS or higher, or any equivalent Linux distribution. Java Development Kit (JDK): JDK Version: Java SE Development Kit 11 or higher (LTS versions recommended). Integrated Development Environment (IDE): Android Studio: Latest ver sion with Android SDK for Android development. IntelliJ IDEA or Eclipse: For Spring Boot and Java backend development. Android SDK: Android SDK with build tools, platform tools, and AVD (Android Virtual Device) Manager for testing. Database: PostgreSQL: Version 12 or higher. It will also need tools like pgAdmin or a similar database management tool for PostgreSQL. Spring Boot Framework: Spring Boot 2.x (compatible with your Java ver sion) for backend development. API Testing Tool: Postman or Insomnia: For testing REST APIs. Authentication: Aadhaar API or Aadhaar Integration Kit (depending on how Aadhaar integration will be implemented). Other Tools: JRE (Java Runtime Environment): Java Runtime Environ 17 ment (JRE) for running the application on machines without development tools installed. Docker (Optional): For containerizing the backend or PostgreSQL database

2. Software Requirements for Mobile Phones (End-User De vices) : Operating System (OS): Android Version: Android 6.0 (Marshmallow) or higher. Storage: The app should require about 100 MB of free storage for in stallation, plus extra space depending on the size of user messages and attachments. Security: The app should use Android's built-in security features like Key store for encrypting sensitive data. Biometric Authentication Libraries (Optional): For integrating fingerprint or face unlock features for enhanced security. Permissions: Internet Access: For syncing messages and user data with the backend. Storage Access: For saving or attaching media files (e.g., images or videos) in the messages. Notification Access: To notify users when actions are completed or Aadhaar credentials are deactivated.

3. Software Requirements for PC/Laptop (End Users) : Browser: Google Chrome: Version 80 or higher. Mozilla Firefox: Version 75 or higher. Microsoft Edge (Chromium-based): Version 80 or higher. Internet Access: A reliable internet connection to interact with the application backend, for message retrieval and synchronization. Desktop Application (Optional): If you develop a desktop-based application in addition to mobile, the following software is needed: Operating System (OS): Windows 10 or higher, macOS 10.14 or higher, Ubuntu 18.04 LTS or higher. 18 Java Runtime Environment (JRE): JRE 11 or higher (if you're deploying a Java-based desktop application). Browser (Optional): Google Chrome or Firefox

IV. SYSTEM ARCHITECTURE



The system architecture depicted in the diagram follows a multi-layered structure, ensuring modularity, security, and scalability. It comprises five main layers: the Frontend Layer, Client-Side Security, Backend Layer, Database Layer, and Server Layer (Nginx). Each of these layers plays a crucial role in the overall functioning of the application.

Frontend Layer

It serves as the user's entry point into the system. It includes the login/signup functionality and message composition interface. It also provides the User Interface (UI) components through which users interact with various features, such as creating, editing, or viewing messages.

Client-Side Security

This layer ensures the safety and privacy of data before it reaches the server. It employs AES encryption to secure messages on the client end and uses SSL/TLS protocols for secure data transmission, protecting the system from man-in-the-middle attacks and unauthorized access during data transfer.

Backend Layer

It handles the core logic of the application. It exposes API endpoints to interact with the frontend and includes various services such as message delivery, user authentication, and message verification. Additionally, backend security mechanisms are implemented to ensure safe processing and communication between components.

Database Layer

It is responsible for storing and managing persistent data. It consists of three main tables: the User Table (for storing user details), the Message Table (for saving composed messages), and the Recipient Management Table (for handling message recipients). This structured storage allows efficient data retrieval and management.

Server Layer

It implemented using Nginx, supports reverse proxy functions, enforces firewall rules for added protection, and provides load balancing to manage high traffic efficiently. It also supports HTTPS to further secure communication between clients and servers.

Together, these layers form a robust, secure, and scalable architecture for an application that handles sensitive user interactions and data transmission.

V. IMPLEMENTATION

1)Class Descriptions

- User : Represents the user of the application. It handles user registration, login, and interaction with the system.
- Attributes: userID: String, Unique identifier for the user. name: String, User’s full name. email: String, User’s email address. aadhar: Aadhar object, User’s Aadhar information.
- Methods: register(): Registers a new user, takes personal details and Aadhar info, and stores them in the system. login(): Authenticates the user based on credentials. inputMessages(): Allows the user to enter their last messages. saveMessages(): Saves the entered messages in the system’s database.

2)Aadhar : Represents the Aadhar credentials of a user.

- Attributes: aadharNumber: String, Aadhar number. status: String, Status of the Aadhar (active or deactivated).
- Methods: deactivate(): Deactivates the user’s Aadhar credentials once the user is confirmed dead.

3)Message : Represents a single message the user wants to send to a recipient.

- I)Attributes: messageID: String, Unique identifier for the message. content: String, The content of the last message. sender: User, The user who created the message. recipient: Recipient, The person who will receive the message.
- II)Methods: setContent(String content): Sets the content of the message. setRecipient(Recipient recipient): Assigns a recipient to the message.

4)Recipient : Represents an individual recipient who will receive the user’s last message.

- Attributes: recipientID: String, Unique identifier for the recipient. name: String, Recipient's name. contactInfo: String, Contact information (e.g., phone number, email). II)Methods: setContactInfo(String info): Sets the recipient's contact information.

5)Death Certificate System

An external system that verifies the user's death and deactivates their Aadhar credentials.

- Methods: issueDeathCertificate(User user): Issues a death certificate for the user once death is confirmed. deactivateAadhar(User user): Deactivates the user's Aadhar credentials.

6)Message Service :

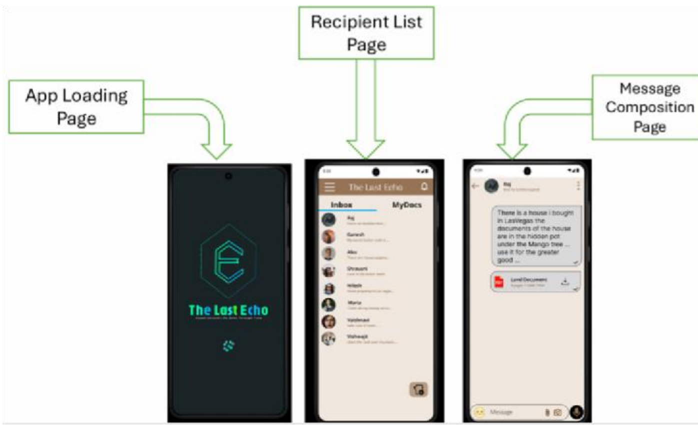
Handles the functionality of saving and forwarding messages.

- Methods: saveMessages(User user, List¡Message messages): Saves a list of messages for the user. forwardMessages(User user): Forwards the user's saved messages to their designated recipients.

7)Method Descriptions

- User.register() Description: Registers a new user by collecting per sonal details and Aadhar information. Parameters: name: String, User's full name. email: String, User's email. aadharNumber: String, Aadhar number. Returns: Boolean, true if registration is successful, otherwise false.
- User.login() Description: Allows the user to log in by validating credentials. Parameters: email: String, User's email address. password: String, User's password. Returns: Boolean, true if login is successful, otherwise false.
- MessageService.saveMessages() Description: Saves the list of last messages entered by the user. Parameters: user: User object, The user for whom the messages are being saved. messages: List¡Message¿, List of messages entered by the user. Returns: void.
- DeathCertificateSystem.issueDeathCertificate() Description: Verifies the user's death and issues a death certificate. Parameters: user: User object, The user whose death certificate is to be issued. Returns: void.
- Aadhar.deactivate() Description: Deactivates the user's Aadhar after their death is confirmed. Parameters: None. Returns: void.

VI. RESULTS



The first image showcases the launch or splash screen of The Last Echo. This screen is the visual starting point of the application, displaying the name and branding of the project. It is designed with a calm, minimalistic aesthetic to reflect the emotional depth of the app's purpose — helping people send their final words to loved ones after their passing. This screen creates the first impression, encouraging a sense of trust and purpose right from the beginning.

Beyond aesthetics, this loading screen can also act as a preparation point for system operations. While this screen is visible to the user, background services such as Aadhaar validation checks, database connectivity, or auto-login sessions can be initialized. It sets the stage for a smooth and thoughtful user journey, offering a clear transition into the more personal and functional parts of the app.

The second image displays the recipient list screen, which is a key part of the message-scheduling process. This page allows users to add, view, or manage the people they want to send their last messages to. It serves as an organized contact list, showing names and possibly quick status indicators for each saved recipient. This interface gives the user control and clarity over who will receive their stored messages, ensuring personalization and trust.

This section also functions as a gateway to the message composition screen for each recipient. Users can tap on any saved contact to proceed to writing a personal note or farewell. The clean layout helps users focus on what really matters — selecting meaningful recipients without unnecessary complexity. This organized list ensures that users can update or edit contacts at any time before the messages are triggered.

The third image captures the core emotional feature of the app — composing a final message. On this screen, the user writes their heartfelt goodbye or important message addressed to a specific person. The design is kept simple to help users focus entirely on their words without distraction. After the message is typed, it's encrypted and saved to the secure database, ready for automated delivery.

This page is arguably the most meaningful part of the app, as it bridges the present with the future. It ensures that, even after someone is no longer alive, their thoughts and feelings can still reach their loved ones. The backend logic ensures that messages stay protected and private, only being triggered when Aadhaar deactivation is detected — making this page the emotional and technical centrepiece of The Last Echo.

VIII. CONCLUSION

The Last Echo represents a powerful and innovative solution for posthumous communication, providing users with the ability to securely convey their final messages to loved ones. By integrating a user-friendly Android interface, a robust backend developed with Spring Boot, and a secure database architecture using PostgreSQL, the project ensures seamless operation and data privacy. The use of Aadhaar for authentication and automated message delivery upon deactivation adds a unique and reliable feature to the application, offering a compassionate service backed by cutting-edge technology. With key components like encryption and database linkage fully integrated, The Last Echo is on track to deliver a user-friendly and dependable platform, providing peace of mind to users knowing their last words will reach their loved ones with care and precision.

Future Scope

1. AI-Powered Message Assistance
   - Integration of Natural Language Processing (NLP) to help users compose meaningful messages.
   - Sentiment analysis to adjust tone based on the recipient's relationship (e.g., spouse, child, friend).
2. Global Identity System Integration
   - Expansion beyond Aadhaar to include international identity systems. o Facilitates global scalability and legal compliance.
3. Legal Document Automation
   - Secure posthumous delivery of documents such as wills, power of attorney, and digital asset keys.
   - Implementation of blockchain for document verification and timestamping.
4. Multimedia Messaging Support
   - Addition of voice and video messages to enhance emotional impact. o Encrypted storage and secure playback features for recipients.
5. Integration with Public Services
   - Partnerships with government death registries, hospitals, and funeral services.
   - Automates Aadhaar deactivation monitoring and message triggering process.
6. Digital Legacy Management
   - Features for social media management after death, such as:
     - Scheduled posts
     - Account deactivation
7. Grief Support and Emotional Wellness Tools

- Integration of resources for grief counseling and mental health support for recipients.
- Periodic remembrance messages or emotional support prompts.

8. Advanced Privacy and Consent Controls
   - Fine-grained controls for message scheduling, recipient permissions, and revocation rights. o Options for backup custodians or legal guardians to manage settings.

X. REFERENCES

1. G. Stringhini and O. Thonnard, ”The EpxDialysis digital Auto Mes sage Forwarding Platform ”, Proceedings of the 2015 International Conference on Detection of Intrusions and Malware and Vulnerability Assessment, pp. 78-97, 2015.
2. Muller, M. Brinkmann, D. Poddebniak, S. Schinzel and J. Schwenk, ” Text messaging to improve retention in hypertension ”, Proceedings of the 2020 IEEE Conference on Communications , pp. 1-9, 2020. C.
3. C. Stransky, O. Wiese, V. Roth, Y. Acar and S. Fahl, ” Automated Messaging After Vehicle Crash ”, Proceedings of the 2022 IEEE
4. Symposium on Security and Privacy, 2022.
5. W. Mayer, A. Zauner, M. Schmiedecker and M. Huber, ” Automatically sends a pre-set emergency message if a wearable health device detects abnormal vital signs ”, Proceedings of the 2016 International Conference on Availability Reliability and Security, pp. 10-20, 2016.
6. H. Hu, P. Peng and G. Wang, ”Auto Message Forwarding To wards Understanding the Adoption of Anti-Spoofing Protocols in Email Systems”, Proceedings of the 2018 IEEE Cybersecurity

Development, pp. 94-101, 201