

**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**NEP CHAT USING CRON SCHEDULING**

**A PROJECT REPORT**

**Submitted To**

**Department of Computer Application**

**Aadim National College**

***In partial fulfillment of the requirements for the Bachelors in Computer Application***

Submitted By:

Bikash Khanal(105902069)

Jestha 2081

6th Semester

Under the supervision of

**Er. Lokesh Gupta**

# Abstract

The development of the real-time chat application was aimed at providing Android users with a seamless and user-friendly messaging platform. The project originated from the growing need for fast, secure, and flexible communication tools that cater to both personal and professional interactions. It specifically targeted the limitations of traditional messaging apps by introducing enhanced features such as scheduled message delivery and the ability to delete messages from both the sender’s and receiver’s ends, giving users more control over their conversations.

The main objective was to design and develop a mobile application that enables users to easily connect and communicate in real time. Additionally, the project aimed to create a clean and intuitive user interface that balances functionality with simplicity.

To achieve these objectives, React Native was used for frontend development, ensuring cross-platform support with near-native performance. The backend was developed using Node.js with Express.js, while MongoDB handled data storage for users and messages. Socket.IO was used to implement real-time communication functionality.

The final application successfully met its goals by delivering reliable real-time messaging, support for scheduled messages, and message deletion features. Testing results confirmed the system’s performance, reliability, and readiness for real-world deployment.

*Keywords:* *Chat application*, *React Native*, *Node.js*, *Socket.IO*, *MongoDB*, *Android users*, *Real-time messaging*, *Cross-platform*

# Acknowledgements

I would like to express my heartfelt gratitude to everyone who contributed to the successful development of Nep Chat using Cron Scheduling.

First, my sincere thanks go to our project supervisor, Er. Lokesh Gupta, for his invaluable guidance, mentorship, and continuous support. His insights played a vital role in shaping the direction and success of this project.

I am also grateful to our respected Coordinator, Er. Prem Subedi, for providing us with essential resources and encouragement throughout the project duration.

My appreciation extends to Aadim National College, especially the Department of IT Department, for fostering a supportive academic environment and providing the platform to execute this project.

I would also like to thank our fellow team members for their dedication and collaboration, and my families and friends for their constant encouragement and belief in us.

This project would not have been possible without the collective support of all these individuals and institutions.

Sincerely,

Bikash Khanal (105902069)

**Table of Contents**

[**Abstract i**](#_Toc199273447)

[**Acknowledgements ii**](#_Toc199273448)

[**List of Figures iv**](#_Toc199273449)

[**List of Tables v**](#_Toc199273450)

[**Chapter 1: Introduction 1**](#_Toc199273451)

[**1.1 Introduction 1**](#_Toc199273452)

[**1.2 Problem Statement 1**](#_Toc199273453)

[**1.3 Objectives 2**](#_Toc199273454)

[**1.4 Scope and Limitation 2**](#_Toc199273455)

[**1.4.1 Scope 2**](#_Toc199273456)

[**1.4.2 Limitations 2**](#_Toc199273457)

[**1.5 Development Methodology 2**](#_Toc199273458)

[**1.6 Report Organization 3**](#_Toc199273459)

[**Chapter 2: Background Study and Literature Review 4**](#_Toc199273460)

[**2.1 Background Study 4**](#_Toc199273461)

[**2.2 Literature Review 4**](#_Toc199273462)

[**Chapter 3: System Analysis and Design 7**](#_Toc199273463)

[**3.1 System Analysis 7**](#_Toc199273464)

[**3.1.1 Requirement Analysis 7**](#_Toc199273465)

[**3.1.2 Feasibility Analysis 8**](#_Toc199273466)

[**3.1.3 Data Modeling (ER Diagram) 9**](#_Toc199273467)

[**3.2 System Design 9**](#_Toc199273468)

[**3.2.1 Architecture Design 9**](#_Toc199273469)

[**3.2.2 Database Schema Design 9**](#_Toc199273470)

[**3.3 Algorithm Details 9**](#_Toc199273471)

[**Chapter 4: Implementation and Testing 11**](#_Toc199273472)

[**4.1 Implementation 11**](#_Toc199273473)

[**4.1.1 Tools Used 11**](#_Toc199273474)

[**Chapter 5: Conclusion and Future Recommendations 13**](#_Toc199273475)

[**5.1 Conclusion 13**](#_Toc199273476)

[**5.2 Lesson Learnt/Outcome 13**](#_Toc199273477)

[**5.3 Future Recommendations 13**](#_Toc199273478)

# List of Figures

[**Figure 1.1 Waterfall Methodology....................................................................................3**](#_Toc199272754)

**Figure 3.1 UsecaseDiagram...............................................................................................**7

**Figure 3.2 Architecture Design..........................................................................................9**

# List of Tables

**Table 3.1 Functional Requirements..................................................................................7**

**Table 3.2 Non-functional Requirements...........................................................................8**

# Chapter 1: Introduction

## Introduction

Nep Chat is an innovative messaging platform designed to provide users with a secure, efficient, and feature-rich communication experience. Its simplicity and versatility make it ideal for individuals, businesses, and communities seeking seamless and secure communication. This app control over personal conversations.

By combining powerful security measures with user-centric innovations, Nep Chat redefines the messaging experience. Its simplicity and versatility make it accessible for everyday users, while its privacy-focused features offer businesses a secure platform for collaboration. Whether you need safe personal messaging, secure business discussions, or a reliable chat platform, Nep Chat provides a seamless and protected communication environment. With its focus on privacy, flexibility, and innovation, Nep Chat is setting a new standard for modern messaging platforms.

## Problem Statement

Without a Chat that addresses key privacy, communication, and users face several challenges. The lack of end-to-end encryption leaves sensitive information vulnerable to breaches. Users may also find it difficult to manage their chat histories, making it harder to protect private conversations.

Some of the major problems are:

* Lack of Privacy: Without strong encryption, conversations are exposed to potential data breaches.
* Connectivity Issues: Without users in areas with poor internet coverage cannot communicate.
* Risk of Data Misuse: Uncontrolled message retention increases the risk of sensitive information being misused.
* Disconnected Conversations: Poor connectivity can cause delays in sending or receiving messages, affecting real-time communication.
* Frustration Over Delays: Delays in sending messages due to poor connectivity lead to frustration and broken conversations.
* No Security Guarantees: Users may hesitate to share confidential information, fearing a lack of protection.

## Objectives

The main objectives of this project can be enumerated as follows:

* To create a user-friendly android platform where users can add others users and can message with each other.
* To integrate features like scheduled messaging enabling users to plan and manage their communication effectively.
* To allow users to delete messages from both ends.

## Scope and Limitation

### Scope

* The project will allow users to send and receive real-time messages securely
* The application will support features such as one-on-one messaging, message scheduling.
* Users can view and manager their respective chat histories.

### Limitations

* There is no push notifications to notify users about new messges and requests
* There is no feature for group chats

## Development Methodology

###### For the methodology of the proposed system, I am using the waterfall methodology.

The **Waterfall methodology** is a traditional, linear approach to software development where the process flows sequentially through distinct phases: requirements gathering, system design, implementation, testing, deployment, and maintenance. Each phase must be completed before moving to the next, and changes are difficult to incorporate once a stage is finalized. It's best suited for projects with clearly defined requirements and minimal expected changes. While it's simple and structured, its rigidity makes it less effective for complex or evolving projects compared to more flexible models like Agile.



**Figure 1.1 Waterfall Methodology**

## Report Organization

The first part of the report contains the summarized introduction of the whole report. It includes the overview, scope and limitation, problem statement and objectives of this project.

The second chapter includes background study i.e., description of fundamental theories, general concepts and terminology related to the project. It also includes the literature review i.e., review of the similar projects, research and theories done by other researchers.

The third chapter includes the system analysis and design phase in which the report of functional and non-functional requirements of the project is stated using use case and system diagrams. It also includes the feasibility study about the system which explains whether the system development process is affordable and within the knowledge range of the developers. It shows the technical, operational and economic feasibility of the project development phase. The explanation of the designing of the system is also done in this chapter. It includes data modeling and process modeling which is explained by using ER diagram and Data Flow Diagram. The architectural design, database design and the user interface design are also listed in this chapter.

# Chapter 2: Background Study and Literature Review

## 2.1 Background Study

In today’s digital era, communication is central to both personal and professional interactions. Traditional methods of communication such as SMS, emails, and phone calls have been supplemented—and in many cases replaced—by instant messaging platforms due to their speed, convenience, and multimedia capabilities.

Chat applications have grown rapidly, driven by the widespread availability of the internet, smartphones, and cloud-based technologies. They enable real-time exchange of messages, voice notes, images, and videos, supporting both individual and group conversations. Popular applications like WhatsApp, Messenger, Slack, and Telegram have demonstrated how messaging tools can enhance collaboration, productivity, and social engagement.

The need for a secure, efficient, and reliable communication platform has led to the development of custom chat applications tailored for specific user bases, such as internal workplace communication, educational environments, or customer support systems. These systems often offer features such as user authentication, contact management, media sharing, and administrative control, and can be extended to include end-to-end encryption and voice/video calling.

This project aims to design and implement a chat application that addresses the need for real-time communication within a defined user community. It focuses on delivering core messaging functionalities along with essential user management and security features.

## 2.2 Literature Review

Chat applications such as WhatsApp, Messenger, Viber, and Instagram have significantly transformed the landscape of digital communication by enabling real-time, multimedia-rich, and secure interactions across vast global user bases. These applications employ a variety of backend technologies, protocols, and user interface strategies to support scalability, responsiveness, and user engagement. WhatsApp, for example, has been a pioneer in integrating end-to-end encryption (E2EE) via the Signal Protocol, ensuring high levels of privacy and security for personal communication [1]. The platform also offers seamless synchronization, cloud backup, and minimal data consumption, making it highly suitable for users in bandwidth-constrained regions [2].

Messenger, another Meta product, enhances the chatting experience by offering smart replies, chatbots, and business integrations through APIs. It utilizes a hybrid approach of long polling and WebSockets to support real-time communication [3]. Its integration with Facebook’s social graph has allowed richer contextual conversations and business interactions. In contrast, Viber focuses on user-centric security and community features, offering functionalities such as self-destructing messages and public group chats. Studies emphasize its regional success in markets like Eastern Europe and Southeast Asia, attributing its growth to localized features and strong privacy protocols [4].

Instagram, while originally centered around visual content, has evolved its messaging system (Instagram Direct) to incorporate chat capabilities that include voice and video calls, disappearing messages, and cross-platform messaging with Facebook Messenger [5]. Its unique integration of ephemeral content and visual media with messaging has influenced communication styles, particularly among younger demographics [6].

Despite the technological advancements and widespread usage of these chat applications, several research gaps persist. First, while many studies have focused on the technical implementation of features such as encryption, scalability, and interface design, there is limited comparative research evaluating the real-world performance and reliability of these systems under high-load, low-bandwidth, or offline conditions. Additionally, although E2EE is a commonly adopted feature, there is insufficient literature evaluating user awareness and understanding of encryption standards across different platforms, which can impact trust and adoption [7].

Moreover, cross-platform messaging (such as between Messenger and Instagram) introduces new challenges related to user privacy, data interoperability, and regulatory compliance, especially in regions governed by strict data protection laws. These areas remain underexplored in current literature. Furthermore, most existing research is application-specific, with a lack of comprehensive comparative analyses that examine how different apps handle similar features such as content moderation, AI-driven recommendations, and user behavior tracking. Finally, while chatbot integration and artificial intelligence have been deployed across several platforms, more empirical studies are needed to assess the effectiveness and user satisfaction of these AI-driven interactions in real-time chat environments.

In conclusion, while applications like WhatsApp, Messenger, Viber, and Instagram have advanced the domain of digital communication significantly, the literature reveals gaps in areas such as performance benchmarking, encryption usability, cross-platform data governance, and AI evaluation. Addressing these gaps is essential to improving the next generation of secure, intelligent, and user-friendly chat applications.

# Chapter 3: System Analysis and Design

## 3.1 System Analysis

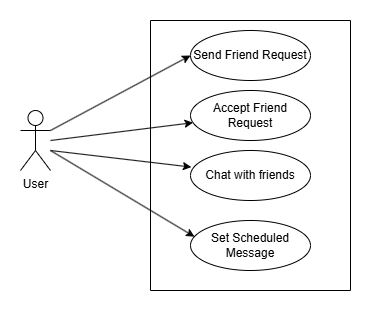
This chapter discusses about the proposed model of the system, the methodology used in building the system, tools and techniques, requirement analysis, requirement specification. It also talks about the functional requirements of the system, the system design. The application architecture of the system will also be shown in this chapter, the use case diagram, data design, activity diagrams, dataflow diagram, control flow diagram, entity-relationship diagram (ERD) and user interface design will all be shown in this chapter.

### 3.1.1 Requirement Analysis

The following section presents the complete set of functional requirements

#### Functional Requirements

Following figure shows the requirements



**Figure 3.1 Usecase Diagram**

**Table 3.1 Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Description** | **Type** |
| R-101 | The android application will contain user interface. | Functional |
| R-102 | The android application will require internet to operate. | Configuration |
| R-103 | The android application will allow navigating menus. | Functional |
| R-104 | The android applications will allow users to add and accept friends. | Functional |
| R-105 | The android application will allow users to chat with each other and set scheduled messages | Functional |

#### Non-functional Requirements

Some of the contents of non-functional requirements are shown table below.

**Table 3.2 Non-functional requirements**

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Description** | **Type** |
| NR-101 | The android application shall ensure sensitive information is secure | Security |
| NR-102 | The android application will be user friendly | Usability |
| NR-103 | Unauthorized users will not be able to access the system. | Security |
| NR-104 | The android application shall run well on desktop and mobile devices | Configuration |

### 3.1.2 Feasibility Analysis

Some important feasibility studies are mentioned below:

#### Technical Feasibility

It is technically feasible as I already have hardware and software required for development of a software. Also, I have technical knowledge on how the project is made through programming language.

#### Operational Feasibility

It is operationally feasible as I am making this system by removing the threats and weakness of existing non manageable system which is reliable for the users.

#### Economic Feasibility

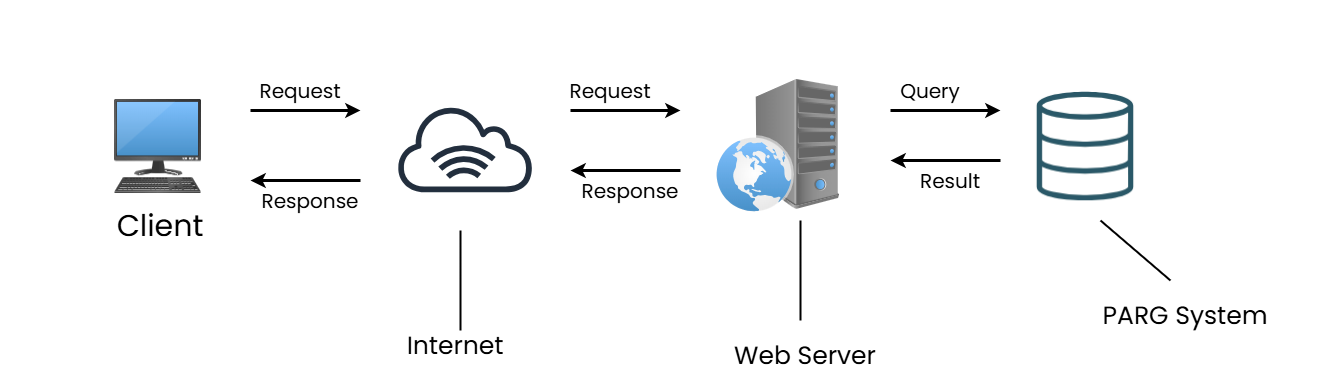
The system does not require extra software and hardware. So, there is no recurring cost than just the internet connection.

### 3.1.3 Data Modeling (ER Diagram)

Since this system uses NoSql Database, there will be no ER Diagram

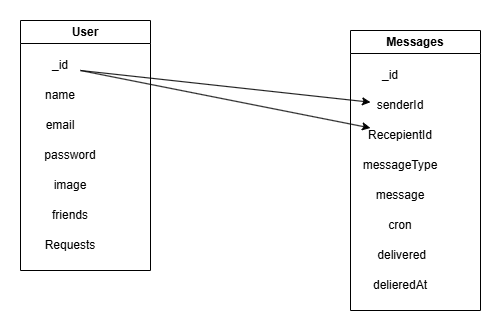
## 3.2 System Design

### 3.2.1 Architecture Design



**Fig 3.2 Architecture Design**

### 3.2.2 Database Schema Design



**Fig 3.3 Database Schema Diagram**

## 3.3 Algorithm Details

A **cron scheduling algorithm** is used to determine the next time(s) a job should run based on a cron expression. Cron expressions are widely used in Unix-based systems (like Linux) and scheduling tools (like Kubernetes, AWS CloudWatch, etc.) to define time-based job execution.

What is a Cron Expression?

A standard cron expression is made up of **5 fields (or 6 with seconds)**, typically in this order:

# ┌───────────── minute (0 - 59)

# │ ┌───────────── hour (0 - 23)

# │ │ ┌───────────── day of the month (1 - 31)

# │ │ │ ┌───────────── month (1 - 12)

# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday=0 or 7)

# │ │ │ │ │

# │ │ │ │ │

# \* \* \* \* \*

Example:

0 14 \* \* 1-5 → Every weekday at 2:00 PM

How the Cron Scheduling Algorithm Works

The cron scheduling algorithm follows these steps:

1. **Parse the Expression**

Break the cron expression into its individual fields (minute, hour, day, etc.), and convert wildcards (\*), ranges (1-5), and lists (1,3,5) into concrete sets of values.

2. **Current Time Initialization**

Start with the current time (now) and incrementally check the next possible time that satisfies all fields in the cron expression.

3. **Field Matching**

From left to right:

* Match minute, if not matched, go to next minute.
* Match hour, if not matched, go to next hour (reset minute).
* Match day of month/week, if not matched, go to next day (reset hour and minute).
* Match month, if not matched, go to next month (reset day, hour, minute).
* Repeat until a valid date-time match is found.

4. **Return Next Matching Time**

Once all fields match, that's your next scheduled run time.

**Code used in project**

# Chapter 4: Implementation and Testing

## 4.1 Implementation

The aim of this chapter is to document the process of development of the main features. It gives a detailed breakdown of the problems encountered and how they were resolved. It also goes through the test plan and test report of the project to ensure all the functionalities are functioning properly. This chapter is where the project is going to be implemented. However, the software and hardware components used in the implementation of this project will be analyzed below.

### 4.1.1 Tools Used

**React Native**

**React Native** is an open-source mobile application framework developed by Facebook. It allows developers to build natively rendered mobile applications for **iOS** and **Android** using **JavaScript** and **React**, a popular JavaScript library for building user interfaces. Unlike traditional hybrid app frameworks, React Native renders components using native APIs, resulting in better performance and user experience. One of the major advantages of React Native is **code reusability**, allowing developers to write a single codebase that works across multiple platforms. It also supports features like hot reloading, modular architecture, and strong community support, making it an ideal choice for mobile app development.

**Node.js and Express.js**

**Node.js** is an open-source, cross-platform, JavaScript runtime environment built on Chrome’s **V8 JavaScript engine**. It is designed for building scalable and efficient server-side applications. Node.js uses an **event-driven, non-blocking I/O model**, making it lightweight and perfect for data-intensive real-time applications.

**Express.js** is a minimalist and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It simplifies the development process by handling common backend functionalities such as routing, middleware integration, request handling, and session management. Express is widely used for creating RESTful APIs, which serve as a communication layer between the client (e.g., a React Native app) and the database.

**MongoDB**

**MongoDB** is a popular **NoSQL** database that stores data in a **flexible, JSON-like format** called BSON (Binary JSON). Unlike traditional relational databases, MongoDB does not require predefined schemas, making it highly adaptable to changes in data structure. This flexibility is particularly useful for applications that deal with unstructured or semi-structured data.

MongoDB is designed for scalability and performance, supporting features like horizontal scaling, replication, and high availability. Its ability to handle large volumes of data and its compatibility with JavaScript-based environments like Node.js make it a preferred choice for modern full-stack applications.

# Chapter 5: Conclusion and Future Recommendations

## 5.1 Conclusion

The chat application developed through this project successfully fulfills the outlined objectives. A user-friendly Android platform was created using React Native, allowing users to add each other and communicate seamlessly through real-time messaging. The integration of scheduled messaging provided users with the flexibility to manage their conversations more effectively, while the delete-for-both feature enhanced control over message privacy and content management. The system demonstrates strong performance and responsiveness, proving the feasibility of combining React Native with Node.js (Express) and MongoDB for building efficient, real-time communication tools.

## 5.2 Lesson Learnt/Outcome

While developing the project, I learned a lot about creating a modular system, which made the code easier to manage and test. I also realized how important it is to organize data efficiently, especially when using MongoDB to handle databases. Connecting the front-end (React Native) and back-end (NodeJs) using REST APIs helped me understand how to sync data smoothly between them. I also gained experience in handling errors and validating user input, which improved the system's reliability.

## 5.3 Future Recommendations

To enhance the functionality and user experience of the chat application, several improvements can be considered for future development. One significant enhancement is the integration of push notifications, which would allow users to receive real-time alerts for new messages even when the application is running in the background or is closed. This would ensure continuous engagement and timely communication. Another valuable addition would be the implementation of group chat functionality, enabling multiple users to participate in a single conversation, making the app more versatile for team discussions and social interactions. Furthermore, introducing an admin dashboard would provide administrative control over user activity, allowing for monitoring, moderation, and management of the platform to ensure security and prevent misuse. Lastly, ensuring cross-platform availability by developing versions of the application for iOS and web browsers would significantly broaden the user base and accessibility, allowing users to interact seamlessly across different devices and operating systems.

**References**

[1] T. Koponen et al., "The Signal Protocol: A Technical Overview," Open Whisper Systems, White Paper, 2016. [Online]. Available: https://signal.org/docs/

[2] R. Singh and A. Srivastava, "Analysis of WhatsApp Messenger's Usage Pattern and Impact on Communication Behavior," *International Journal of Scientific Research in Computer Science*, vol. 6, no. 1, pp. 120–124, Jan. 2018.

[3] S. H. Ko and J. Kim, "Design and Implementation of a Real-Time Messaging System Based on WebSockets," in *Proc. IEEE Int. Conf. Consumer Electronics*, Las Vegas, NV, USA, 2019, pp. 1–4.

[4] D. Kalabokis and A. Antoulas, "Secure Messaging and the Evolution of Viber as a Multi-Purpose Chat Application," *International Journal of Communication Networks and Information Security*, vol. 13, no. 2, pp. 245–251, June 2021.

[5] D. Rosenberg, "Behind the Scenes: Unifying Messenger and Instagram Messaging," *Facebook Engineering Blog*, Sept. 2020. [Online]. Available: https://engineering.fb.com

[6] E. Marwick, "The Public Domain: Social Media, Privacy, and Ephemerality," *Journal of Broadcasting & Electronic Media*, vol. 61, no. 3, pp. 528–537, Sept. 2017.

[7] P. J. Denning and C. Martell, "Understanding Security and Privacy in Chat Applications: A Human-Centered Perspective," *IEEE Security & Privacy*, vol. 19, no. 1, pp. 22–29, Jan.–Feb. 2021.