A

Project Report

On

Pulse Messaging Engine

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Navneet Pathak	**2261385**
Deepak Singh Chamyal	**2261173**
Tanuja Arya	**2264003**
Priyanshu Joshi	**2261442**

Under the Guidance of

Mr. Ansh Dhingra

Lecturer

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT’S DECLARATION

We, **Navneet Pathak, Deepak Singh Chamyal, Tanuja Arya, Priyanshu Joshi** hereby declare the work, which is being presented in the project, entitled ‘Pulse Messaging Engine’ in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Ansh Dhingra.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Navneet Pathak

Deepak Singh Chamyal

Tanuja Arya

Priyanshu Joshi

CERTIFICATE

The project report entitled “Pulse Messaging Engine” being submitted by Navneet Pathak S/o Gokula Nand Pathak (2261385), Deepak Singh Chamyal S/o Khimpal Singh Chamyal (2261173), Tanuja Arya d/o Mr. Mahendra Lal Arya (2264003), Priyanshu Joshi S/o Rajendra Prasad Joshi (2261442) of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Ansh Dhingra
(Project Guide)

Dr. Ankur Singh Bisht
(Head, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Pulse Messaging Engine'** (Lecturer, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Navneet Pathak (2261385)
Deepak Singh Chamyal (2261173)
Tanuja Arya (2264003)
Priyanshu Joshi (2261442)

Abstract

Pulse Messaging Engine is a real-time, full-stack messaging application designed to facilitate secure and instantaneous communication between users, integrated with a payment system. The project aims to provide a seamless user experience that combines live chat, user presence, and digital transactions within a single platform.

Built using modern technologies like **React.js** on the frontend and **Node.js with Express.js** on the backend, the system leverages **MongoDB** for data persistence and **Socket.IO** for real-time communication. The application features user authentication via **JWT tokens**, secure password handling using **bcryptjs**, and real-time balance transactions with atomic operations supported by MongoDB.

Users can chat, manage profiles, transfer funds securely, and receive instant notifications. The frontend incorporates **Zustand** for state management and **Tailwind CSS** with **DaisyUI** for responsive and aesthetic UI design. File uploads such as profile pictures are handled using **Cloudinary**.

The Pulse Messaging Engine is a scalable, extensible, and user-friendly platform designed to meet modern demands for fast communication and integrated financial interactions. Future enhancements include support for media sharing, group chats, end-to-end encryption, and voice/video calling capabilities.

TABLE OF CONTENTS

List of Abbreviations – Pulse Messaging Engine

1. API: Application Programming Interface – A set of functions enabling interaction between frontend and backend services.
2. ADB: Application Debug Bridge – (Contextual redefinition) A tool or mechanism used for inspecting real-time logs and debugging issues in the Pulse system (e.g., Socket.IO logs or API request logs).
3. ART: Asynchronous Real-Time – Represents the non-blocking real-time event-handling mechanism used in Pulse via WebSockets.
4. Baseband: In a real-time messaging context, refers metaphorically to the “base” data signals (like raw message payloads) before encoding or formatting for transport (e.g., via WebSocket).
5. Binary (BIN): A compiled executable or deployment-ready file. In Pulse, this can refer to the backend service builds or Vite-generated frontend bundles.
6. JWT: JSON Web Token – A secure token format used for user authentication and session management.
7. VM: Virtual Machine – Refers to runtime environments like Node.js or cloud server instances used to run the backend services.
8. DB: Database – MongoDB used for storing messages, users, and transaction data.
9. WS: WebSocket – A communication protocol used in Pulse for real-time chat and payment alerts.
10. SDK: Software Development Kit – Tools or libraries (e.g., Cloudinary SDK, JWT library) integrated into Pulse to extend its capabilities.
11. UI/UX: User Interface / User Experience – The design and interactive components that form the front end of Pulse.
12. P2P: Peer-to-Peer – Direct interaction model used in chat and payment flows between users.

1 INTRODUCTION

1.1 Prologue

Welcome to the official documentation of **Pulse Messaging Engine**, a modern, real-time messaging platform with integrated peer-to-peer payment functionality. This project represents the convergence of seamless communication and secure transaction capabilities, built to address the needs of fast-paced digital interactions in today's connected world.

1.2 Background and Motivations

The rise of real-time communication and digital payment systems has transformed how people connect and exchange value. Traditional chat applications often lack financial integration, while most payment systems do not provide conversational context. This gap inspired the development of **Pulse Messaging Engine** — a platform where users can chat and perform secure monetary transactions without switching apps.

The motivation behind Pulse is to create a unified ecosystem where communication and financial exchange coexist naturally, enabling faster collaboration, micro-transactions, and a better user experience, especially in environments like freelancing, peer-to-peer sales, or collaborative digital communities.

1.3 Problem Statement

Current messaging applications fall short when it comes to integrating secure and real-time payments within the conversation flow. Users are forced to switch between multiple apps to communicate and transact, which disrupts user experience and increases the risk of error and fraud. Moreover, many real-time apps suffer from poor scalability, lack of strong security practices, or outdated design systems.

The **Pulse Messaging Engine** seeks to solve this problem by combining a robust, Socket.IO-powered messaging system with a secure, transaction-based payment system, all embedded into a single, responsive user interface.

1.4 Objectives and Research Methodology

Objectives:

- To design and implement a full-stack web application that supports real-time messaging and payments.
- To provide a secure authentication and authorization mechanism using JWT and HTTP-only cookies.
- To implement atomic payment transactions using MongoDB and passkey verification.
- To deliver instant payment and message notifications using WebSockets.
- To create a responsive, modern UI using React, Tailwind CSS, and Zustand for global state management.

Research Methodology:

- **Requirement Analysis**: Identified gaps in existing platforms and user needs.
- **Technology Review**: Studied modern full-stack technologies including Node.js, MongoDB, React, and WebSockets.
- **Prototype Development**: Built modular prototypes to validate core concepts such as chat delivery and secure transaction flow.
- **Iterative Implementation**: Applied Agile principles, developing the project in iterative stages with continuous testing and feedback.
- **Testing & Evaluation**: Performed functional testing on modules like authentication, messaging, and payment to ensure robustness and security.

1.5 Project Organization

The Pulse Messaging Engine project is organized into two main components: **Frontend** and **Backend**, each with clearly defined modules and responsibilities:

- **Frontend**: Built with React.js and Vite, it handles user interface, routing, real-time client interactions (via Socket.IO client), and global state management using Zustand.
    - Components: UI Elements, Chat Interface, Payment Panel, Notifications
    - Pages: Login, Register, Dashboard, Profile, Chat Window

    - Styling: Tailwind CSS and DaisyUI for theming and responsiveness

- **Backend**: Built with Node.js and Express.js, it manages API endpoints, authentication (JWT), database operations (MongoDB with Mongoose), and real-time server connections (Socket.IO).
    - Modules: Authentication, User Management, Messaging, Transactions, WebSocket Events
    - Security: bcryptjs for password hashing, token-based authentication, and secure cookies
- **Database**: MongoDB stores user data, chat history, and transaction records. Transactions use atomic operations to ensure data consistency and security.
- **Real-Time Layer**: Socket.IO enables bidirectional communication between clients and server, handling messaging, typing indicators, online status, and payment alerts.

This modular and layered architecture ensures maintainability, scalability, and a rich user experience.

2 PHASES OF SOFTWARE DEVELOPMENT CYCLE

Hardware and Software Requirements

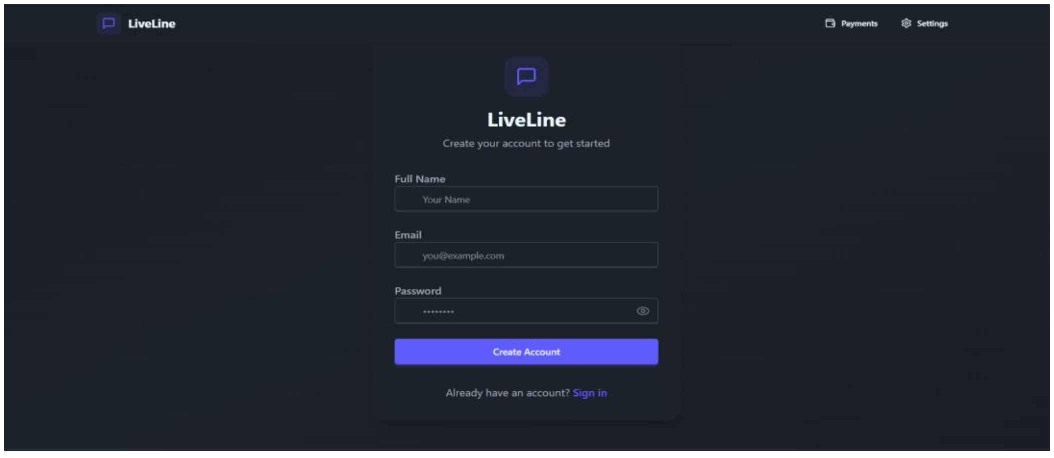
2.1Hardware Requirement

Windows	OS X	Linux
Windows 10 or 11 (64-bit)	macOS Big Sur (11) or later	Ubuntu 20.04 LTS or later
Minimum 8 GB RAM (16 GB recommended)	Minimum 8 GB RAM (16 GB recommended)	Minimum 8 GB RAM (16 GB recommended)
400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches	400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches	400 MB hard disk space plus at least 1 GB for Android SDK, emulator system images, and caches
Intel i5 or AMD Ryzen equivalent processor	Apple M1/M2 or Intel-based Macs supported	Intel i5 or AMD processor
At least 2 GB available disk space	At least 2 GB available disk space	At least 2 GB available disk space

2.2 Software Requirement

I. Visual Studio Code (VS Code)
II. Git
III. Postman
IV. MongoDB
V. Node.js and npm

3 SNAPSHOTS



LiveLine

Create your account to get started

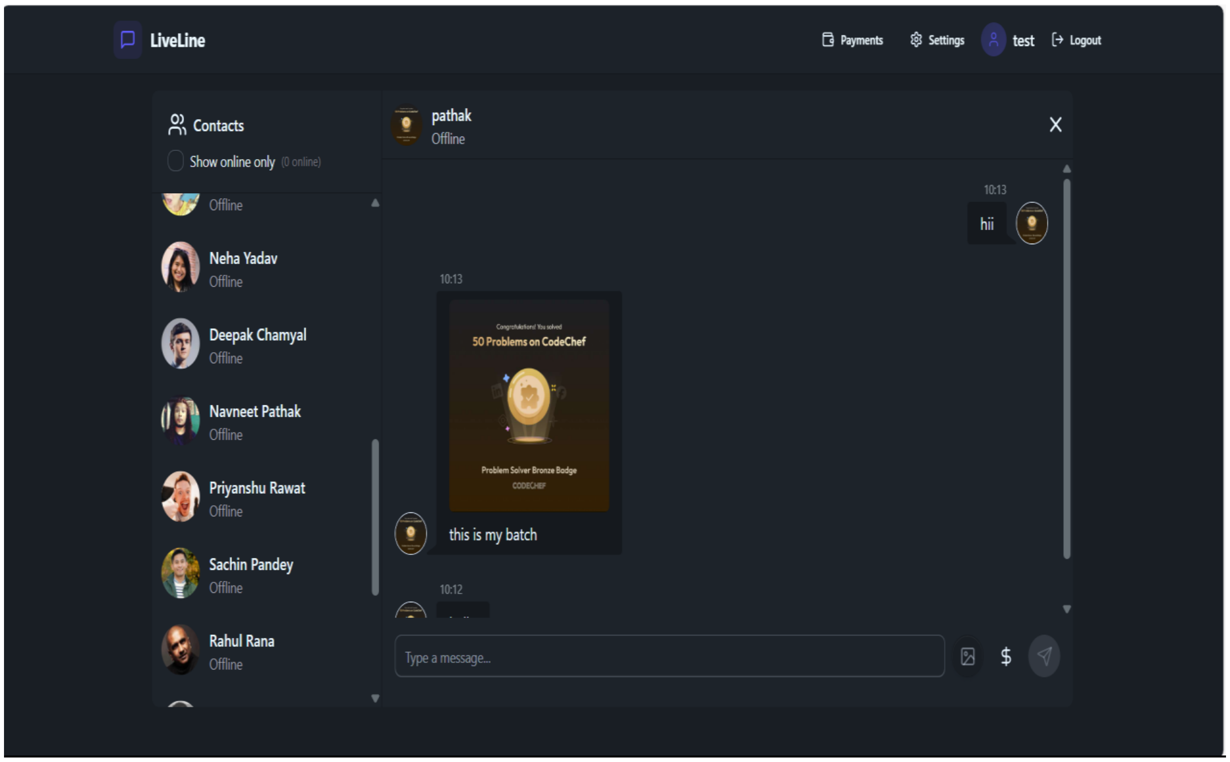
Full Name: Your Name

Email: you@example.com

Password: ••••••••

Create Account

Already have an account? Sign in



Offline

Neha Yadav
Offline

Deepak Chamyal
Offline

Navneet Pathak
Offline

Priyanshu Rawat
Offline

Sachin Pandey
Offline

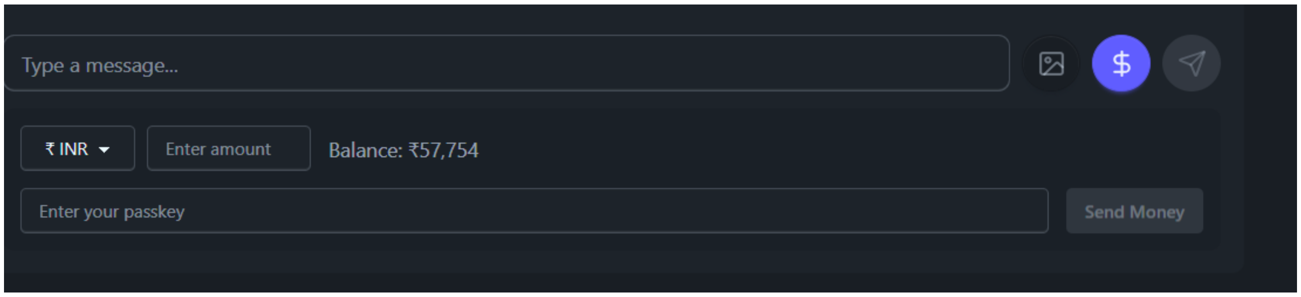
Rahul Rana
Offline

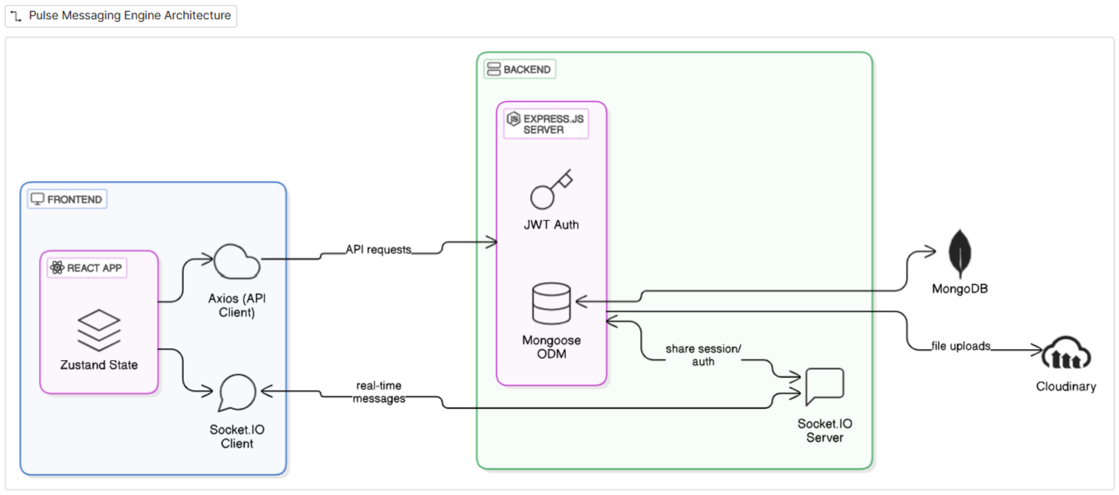
pathak
Offline

10:13
hii

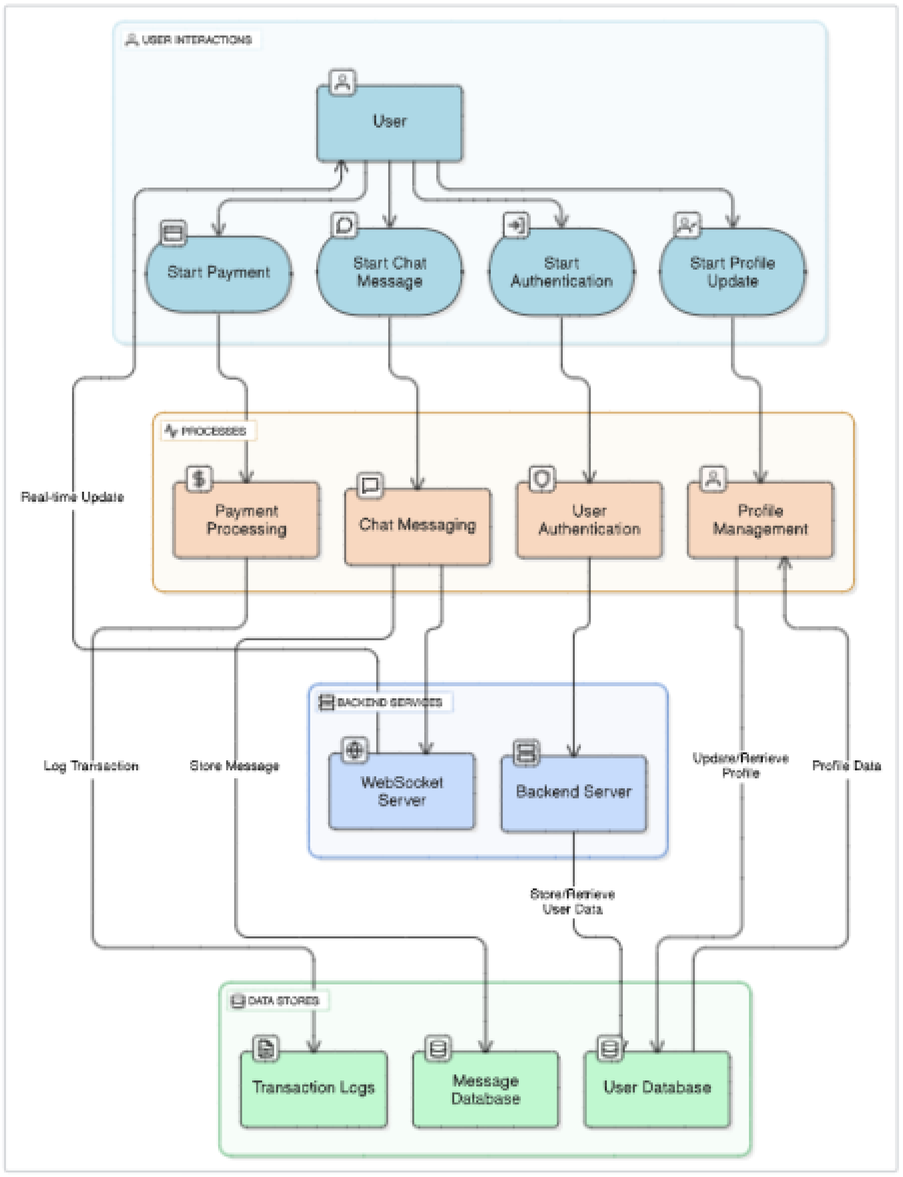
10:13
Congratulations! You solved
50 Problems on CodeChef
Problem Solver Bronze Badge
CODECHEF
this is my batch

10:12


Type a message...
$
₹ INR
Enter amount
Balance: ₹57,754
Enter your passkey
Send Money


Pulse Messaging Engine Architecture
FRONTEND
REACT APP
Zustand State
Axios (API Client)
Socket.IO Client
API requests
real-time messages
BACKEND
EXPRESS.JS SERVER
JWT Auth
Mongoose ODM
share session/ auth
Socket.IO Server
MongoDB
file uploads
Cloudinary

Pulse Messaging Engine - Level 1 Flow Chart



4. LIMITATIONS

There are certain limitations that exist in this project current form:

4.1 Lack of End-to-End Encryption (E2EE)

- Messages are transmitted over Socket.IO and stored in plain text in MongoDB.
- There is no message-level encryption implemented, which limits privacy in highly secure communication environments.

4.2 No Group Chat Functionality

- The system currently supports only one-to-one messaging.
- Group chats, broadcast messages, and chat rooms are not supported yet.

4.3 Basic Payment Security

- Payment authentication relies on a static passkey and JWT-based user verification.
- It lacks advanced financial-grade security protocols such as:
  - OTP verification
  - Biometric integration
  - Real-time fraud detection
  - Role-based authorization.

4.4 No Offline Message Queue

- If the receiver is offline, messages or payments are stored in the database but no queueing or retry mechanism is in place to ensure delivery once the user reconnects.

4.5 Limited Cross-Platform Compatibility

- The frontend is optimized for modern desktop and mobile browsers.
- Native mobile apps (Android/iOS) are not yet supported.

4.6 No Media/File Sharing

- Only text-based communication is allowed.
- There is no current support for image, video, audio, or document transfer in chat.

4.7 Simplified Notification System

- Real-time notifications use Socket.IO but do not support:
    - Persistent push notifications
    - Email/SMS alerts
    - Notification history

4.8 Minimal Logging and Auditing

- The system lacks comprehensive logging for:
    - Payment audit trails
    - Chat activity history
    - Admin-level monitoring and alerting

4.9 Language and Currency Limitation

- Interface is in English only.
- Currency support is limited to INR, USD, and EUR without dynamic exchange rate integration.

4.10 No Role-Based Access Control (RBAC)

- All users have equal access; there is no separation between regular users, moderators, or admins in terms of access or control.

5. ENHANCEMENTS

The Pulse Messaging Engine is designed with scalability and extensibility in mind. Future enhancements aim to enrich the platform's functionality, improve user experience, and strengthen security. Below are the planned and potential future improvements:

5.1 Group Chat and Broadcast Support

- Implementation of group chat rooms and the ability to send broadcast messages to multiple users simultaneously.

5. 2 End-to-End Encryption

- Introduce **E2EE** for chat messages to ensure secure communication even in hostile environments.
- Encryption will happen on the client-side, and messages will only be decrypted by the recipient.

5.3 File and Media Sharing

- Allow users to send and receive images, audio files, videos, and PDFs.
- Integrate with Cloudinary or similar services for efficient media storage and delivery.

5.4 Enhanced Payment Features

- Add **payment receipts**, **transaction history**, and **multi-step authentication** for transactions.
- Integrate with UPI, Stripe, or PayPal APIs for real-world payments.

5.5 Push Notifications

- Implement browser and mobile **push notifications** for messages and payments, even when the app is closed.

5.6 Mobile App Development

- Develop **native Android and iOS apps** for better performance and offline access.
- Possibly use Flutter or React Native for cross-platform support.

5.7 Admin Dashboard

- Introduce a comprehensive admin panel for:

    - User management
    - Monitoring payments and usage analytics

5.8 Chat Search and Filters

- Allow users to search chat history by keyword, date, or user.
- Add filtering options to quickly navigate through messages and transactions.

5.9 Multilingual Interface

- Enable support for multiple languages for a broader user base.
- Integrate language selection and automatic translation capabilities.

6. CONCLUSION

The **Pulse Messaging Engine**, also known as **GEHU Question World**, is a real-time academic support platform developed to assist students of Graphic Era Hill University in preparing for their regular and back examinations. This project integrates modern web technologies to deliver a seamless and secure environment where students can access and exchange previous year question papers efficiently.

By combining instant messaging capabilities with academic resource sharing, the platform ensures that users are always connected and have access to essential study materials. Its simple interface, secure authentication, and real-time updates make it a practical solution for both academic collaboration and communication.

Looking toward the future, the application is designed to scale with added functionality. One of the most impactful enhancements will be the inclusion of **answers alongside question papers**, making it an all-in-one solution for examination preparation.

This project addresses a critical gap in academic resource accessibility and paves the way for a smarter, more connected educational ecosystem at GEHU.

REFERENCES

1. Express.js Documentation.
2. MongoDB Documentation.
3. Socket.IO Documentation.
4. React.js Official Guide.
5. Tailwind CSS Documentation.
6. Cloudinary API Reference.