PROJECT REPORT

ON

CHATTING APPLICATION

SUBMITTED TO

J.C Bose University of Science and Technology, Faridabad

IN THE PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

Computer Science and Engineering

Submitted To

Ms. Jyoti Kashyap

(Assistant Professor)

Submitted By

Raju Kumar

(22011004507)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARAVALI COLLEGE OF ENGINEERING AND MANAGEMENT, FARIDABAD - 121002

(JUNE-2025)

TRAINING LETTER

NEWTAB TECHNOLOGIES

Certificate of Internship Completion

Newtab Technologies
Kukatpally, Hyderabad, India,
500085
info@newtabtechnologies.com

This is to certify that **Raju Kumar** has completed an internship at **Newtab Technologies** as a **Junior Java Backend Developer Intern** for the duration of **January 2025** to **June 2025**

During this period, he has been actively involved in various stages of backend development using Java and related technologies. He demonstrated a keen interest in learning modern backend engineering practices and made valuable contributions to ongoing projects.

Key responsibilities included:
- Assisting in designing and implementing RESTful APIs using Java (Spring Boot)
- Collaborating with frontend and database teams to ensure seamless integration
- Debugging, testing, and maintaining backend codebases following best practices
- Participating in daily standups and code review sessions

Raju Kumar consistently demonstrated professionalism and a strong willingness to learn throughout the internship. We believe he will be an asset to any future team he joins.
We wish him the very best in all future endeavors.

Issued on: July 01, 2025

Warm regards,

Venkat Sai Kolli
Founder and UI/UX Designer,
Newtab Technologies.

info@newtabtechnologies.com www.newtabtechnologies.com

DECLARATION

I hereby declare that the project work entitled "Chatting Application" submitted to **JC Bose University of Science and Technology Faridabad**, Haryana (India), is a record of an original work done by us under the guidance of Mrs. Joyti (Assistant Professor) in Computer Science and Engineering, **ARAVALI COLLEGE OF ENGINEERING AND MANAGEMENT, FARIDABAD**, and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING.

Dated **Student Name**

Raju Kumar

ACKNOWLEDGEMENT

This project would not have taken shape without the guidance provided by **Ms. Jytoti Kashyap** Trainer who helped in the modules of our project and resolved all the technical as well as other problems related to the project and, for always providing us with a helping hand whenever we faced any bottlenecks, in spite of being quite busy with their hectic schedules. Above all we wish to express our heartfelt gratitude to **Ms. Sakshi Kumar**, H.O.D, CSE DEPARTMENT whose support has greatly boosted our self-confidence and will go a long way in helping us to reach further milestones and greater heights.

HOD **Mentor**

Ms. Sakshi Kumar **Ms. Jyoti Kashyap**

TABLE OF CONTENTS

COMPANY PROFILE

Newtab Technologies is a technology-driven incubation platform designed to accelerate the growth of early to mid-stage startups through its structured Virtual Incubation Program. The organization leverages a digital-first framework to deliver incubation services remotely, thereby eliminating geographical limitations and ensuring accessibility for entrepreneurs across diverse regions. The program provides startups with a standardized Startup Kit encompassing business model development tools, legal templates, branding resources, and investor pitch support. Startups admitted to the program benefit from domain-specific mentorship and are integrated into a curated network of stakeholders, including venture capitalists, angel investors, accelerator partners, and subject matter experts. The selection process is methodical, consisting of an initial screening by regional leads followed by a secondary evaluation conducted by an internal and external committee. This two-tiered evaluation ensures alignment between startup potential and investor-readiness. The monthly intake model adopted by **Newtab Technologies** further optimizes resource allocation and ensures a streamlined onboarding experience for selected ventures.

Newtab Technologies' ecosystem-centric approach emphasizes measurable growth outcomes, investor engagement, and scalability potential. Startups undergo periodic performance assessments and receive tailored support based on their stage, sector, and traction metrics. The platform encourages cross-collaboration between founders, investors, and mentors through structured demo days, pitch sessions, and knowledge-sharing webinars. **Newtab Technologies** has incubated startups across a wide range of sectors, including financial technology, education technology, health technology, SaaS, and consumer internet, reinforcing its position as a key enabler within the innovation and startup landscape. By integrating incubation best practices with a technology-oriented delivery model, **Newtab Technologies** continues to contribute to the advancement of entrepreneurial innovation on both national and international fronts.

A notable feature of **Newtab Technologies**' model is its emphasis on digital transformation within the startup incubation lifecycle. The platform utilizes proprietary dashboards, real-time tracking systems, and data analytics to monitor the performance of startups and adjust support strategies dynamically. This data-driven approach ensures that each startup receives actionable insights and timely interventions, enhancing the chances of product-market fit and long-term sustainability. Moreover, the use of virtual collaboration tools enables seamless communication between cohort members, mentors, and program managers, ensuring continuity of support irrespective of geographic dispersion.

From an academic and engineering standpoint, **Newtab Technologies** represents a scalable, modular framework for virtual incubation that can be replicated in other contexts. Its ability to combine technical infrastructure with human capital support illustrates an efficient model for fostering innovation in a resource-constrained environment. As technology continues to reshape traditional business models, platforms like **Newtab Technologies** serve as critical infrastructure in promoting inclusive entrepreneurship, democratizing access to growth opportunities, and contributing meaningfully to the knowledge economy.

📍 **Address:**Kukatpally,Hyderabad,India-500085
📧 **Email:** info@newtabtechnologies.com

CHAPTER NO. 1

1. INTRODUCTION

1.1 Project Summary

In the evolving landscape of communication and connectivity, real-time messaging plays a critical role in ensuring instant information exchange, collaboration, and interactive engagement. The Chatting Application is a Java-based desktop application developed to facilitate direct communication between two users through a socket-based client-server architecture. This system is designed to demonstrate real-time data transfer using core Java libraries and GUI components within a controlled local environment.

The primary objective of the Chatting Application is to provide a simple and intuitive platform where two users—acting as client and server—can exchange messages in real time. The application is built using Java Swing for the graphical user interface and Java Socket Programming for managing network communication. It establishes a direct connection between the server and client using TCP sockets, allowing the seamless transmission of text messages. The interface features message formatting, timestamps, and structured layout to enhance readability and user experience.

This system addresses basic challenges in peer-to-peer communication by implementing real-time data flow using input and output data streams, with UI elements that update dynamically during conversation. The project includes core components such as socket initialization, data transmission handling through Data Input Stream and Data Output Stream, layout management with JPanel and BoxLayout, and GUI interaction using JButton, JTextField, and JLabel. The backend logic is implemented using standard Java classes, without reliance on third-party frameworks or databases, promoting a lightweight and focused learning model for understanding network programming.

The Chatting Application exemplifies the practical application of Java in building communication tools and serves as a foundational project for expanding into multi-user chat systems, encrypted messaging, or web-based real-time communication platforms in future iterations.

1.2 Project Objective

The primary objective of the Chatting Application is to build a lightweight, responsive, and user-friendly desktop application that enables real-time two-way communication between users over a network using Java's built-in socket programming capabilities. The system is designed to demonstrate the practical implementation of networking, GUI development, and data stream handling in Java, providing a foundational model for scalable chat-based communication tools.

The key objectives of the project are as follows:

1. **Socket-Based Communication:**

   To establish a reliable client-server connection using Java Sockets (Socket, ServerSocket) for enabling real-time, bidirectional message transfer over a TCP/IP network.

2. **Interactive GUI with Java Swing:**

   To design an intuitive graphical user interface (GUI) using Java Swing components (JPanel, JButton, JTextField, JLabel) that ensures smooth user interaction and visually structured chat display.

3. **Real-Time Messaging:**

   To implement real-time message sending and receiving capabilities using DataInputStream and DataOutputStream, ensuring immediate feedback during conversations.

4. **Time-Stamped Message Display:**

   To design an intuitive admin panel for managing partner profiles, reviewing uploaded documents, monitoring offers, and tracking system activities.

5. **Client and Server Modules:**

   To modularize the system into clearly defined client and server applications that operate independently while maintaining synchronized communication.

6. **User Interaction Handling:**

   To incorporate responsive event listeners and action handlers for seamless user input, message submission, and UI updates during active chat sessions.

7. **Minimal Resource Dependency:**

   To develop a standalone desktop-based system with no reliance on third-party frameworks or external databases, ensuring simplicity and portability.

8. **Foundational Model for Future Expansion:**

   To provide a base architecture that can be extended in future versions to support features like multi-user chat, message encryption, media sharing, or persistent chat history.

1.3 Tools/ Platform Hardware & Software Requirements

The **Chatting Application** is developed using core Java technologies with a lightweight desktop-based architecture, focusing on simplicity, real-time communication, and modular design. The following tools, platforms, and system requirements were used during development:

Hardware Requirements

Component	**Minimum Specification**
Processor	Intel Core i3 / AMD Ryzen 5 or higher
RAM	2 GB (4 GB recommended for smooth build/run)
Storage	256 GB SSD or 500 GB HDD
Display	13" or larger monitor (Full HD preferred)
Internet Connection	Required for version control, package installations, and email notifications

Software Requirements

Software	**Version / Description**
Operating System	Windows 10 or higher
Visual Studio	2022 with .NET Core SDK installed
Java	JDK 8
Microsoft SQL Server	2019 or later
SQL Server Management Studio (SSMS)	Latest version
Git	Latest version
Web Browser	Google Chrome / Microsoft Edge (latest)

1.4 Project Category

The Chatting Application falls under the category of **Desktop-Based Networking Applications**. It is primarily focused on demonstrating core concepts of **real-time communication**, **network socket programming**, and **graphical user interface (GUI)** development using Java. This project emphasizes peer-to-peer interaction over a TCP/IP connection and is suitable for academic demonstration, internal communication systems, and foundational learning of Java-based client-server architecture.

This project aligns with the following domains:

- **Computer Networks**
- **Socket Programming**
- **Java Desktop Application Development**
- **Client-Server Architecture**
- **Real-Time Messaging Systems**

CHAPTER NO. 2

2. SYSTEM STUDY

System Study is the foundational phase of the Chatting Application development, focused on understanding existing communication limitations, identifying technical and operational challenges, and defining system requirements. It begins with analysing the traditional method of communication between individuals, which often lacks immediacy, system integration, or a user-friendly digital platform for one-to-one messaging. The study identifies the primary objective—to develop a lightweight, real-time, Java-based desktop application that enables instant, secure, and interactive communication using socket programming.

Through conceptual understanding and simulation of client-server interaction, the system was designed with clear focus on GUI responsiveness, network reliability, and structured message flow. Functional and non-functional requirements were gathered based on real-world peer-to-peer messaging scenarios. A comprehensive feasibility analysis (technical, operational, and economic) confirmed the viability of this project. Key challenges identified include lack of message synchronization in basic console apps, non-GUI-based implementations, and absence of timestamped interaction. The proposed solution leverages Java's Swing and Networking libraries to provide a fully interactive chat application using a graphical interface. This study lays the groundwork for building a modular, scalable, and user-friendly communication system that aligns with academic and real-world software design practices.

2.1 Preliminary Investigation

The preliminary investigation phase is crucial in understanding the communication gaps in current small-scale or local network-based messaging systems. Many existing Java-based communication tools are console-driven or lack GUI integration, which makes them less usable for non-technical users. Furthermore, console-based systems fail to provide a modern user experience, timestamps, and consistent UI updates, making real-time communication ineffective.

A detailed study was conducted to identify user requirements for one-on-one desktop messaging. It was observed that there is a need for an application that supports live text exchange, proper formatting, timestamping, and structured layout while remaining easy to deploy and maintain. A major concern in basic chat implementations is the lack of proper data encapsulation, inconsistent GUI responsiveness, and poor error handling in network I/O streams.

Additionally, the study highlighted the lack of separation between client and server responsibilities in many basic examples, creating challenges in scaling or extending the system. It became clear that a dual-application setup—client and server—was necessary to simulate a real-world chat system. The proposed Chatting Application overcomes these shortcomings by using socket programming for communication, Swing components for UI, and standard Java I/O streams to maintain message integrity.

2.2 System Development Life Cycle

The System Development Life Cycle (SDLC) provides a structured approach for building software in defined stages. The following SDLC phases were followed in developing the Chatting Application:

1. Requirement Gathering and Analysis

This phase involved identifying the key features expected in a chat system such as real-time text messaging, GUI-based input/output, timestamped messages, and network communication over localhost. User expectations were documented to ensure the application remained lightweight yet functional.

2. System Design

The architecture was designed using a client-server model over TCP/IP sockets. The GUI was structured using Java Swing with JTextField for input, JButton for message sending, and JPanel layouts for message rendering. A vertical BoxLayout and timestamp formatting using SimpleDateFormat were integrated. Socket creation, input/output stream management, and multi-panel layout logic were defined.

3. Implementation

Implementation involved coding both the Client.java and Server.java components. The socket connection, data stream setup, action event handling, UI drawing, and message rendering were all implemented using core Java. The program was tested in loopback (127.0.0.1) mode to simulate two-way communication on a single machine.

4. Testing
Functional testing was conducted to verify input responsiveness, network stability, UI behaviour on message transfer, and timestamp generation. Various edge cases such as socket drop, message flooding, and GUI lag were tested and refined. The client and server were validated independently and together.

2.3 Feasibility Study
A feasibility study was conducted to evaluate the potential for successful deployment and learning benefits of the Chatting Application. The study examined various key aspects:

2.3.1 Technical Feasibility
The technical feasibility evaluates the application's compatibility with existing infrastructure and technology resources. It confirms that the application can be developed using freely available tools such as:

- JDK 8 or above
- Java Swing & Socket libraries
- Standard IDEs like Eclipse or IntelliJ IDEA

The system architecture is capable of supporting client-server communication, with a GUI that operates independently of the backend logic. It also ensures consistent I/O stream behavior and platform-independent execution. The modular Java codebase promotes easy maintenance and debugging. All development tools are readily available and no external dependencies are required.

2.3.2 Operational Feasibility
Operational feasibility assesses how effectively the system can be adopted and used by its intended users. The GUI is designed to be user-friendly, even for those unfamiliar with Java programming. Minimal training is required, and the system can be run on any standard desktop setup with the Java Runtime Environment installed.
The real-time chat interface, along with message timestamping and auto-alignment, ensures that the application mimics real-world messaging behaviour. The operations involved—starting a server and then a client—are simple and consistent with general desktop app use. As the system runs offline (localhost), no server hosting or internet connection is required.

2.3.3 Economic Feasibility
The project is economically feasible due to its reliance on free and open-source tools. Development, testing, and deployment were conducted using:

- Free IDEs (Eclipse/IntelliJ)
- Open-source Java SDK
- No database or licensing costs

There are no hidden costs associated with third-party APIs, web hosting, or maintenance. Since this project is academic in nature and runs on local machines, the cost-benefit ratio is highly favourable. Additionally, the system promotes learning and skill development in GUI programming and network systems at zero cost.

CHAPTER NO. 3

3. SYSTEM ANALYSIS

SYSTEM ANALYSIS

System Analysis is a critical phase in the development of the Chatting Application that involves examining communication needs and translating them into technical specifications. It focuses on understanding the system requirements, identifying communication inefficiencies, and determining the best architectural approach to fulfill real-time messaging between client and server. The system analysis phase typically involves the following steps:

1. **Requirements Gathering:**

   This step involves collecting detailed information about how users intend to communicate. Both functional requirements (e.g., real-time text messaging, timestamped delivery, GUI interface) and non-functional requirements (e.g., responsiveness, security, low resource usage) are analyzed. Stakeholder expectations were simulated in peer-to-peer desktop communication scenarios to define exact goals.

2. **Feasibility Study:**

   Based on the gathered requirements, a feasibility study (technical, operational, economic) was conducted to ensure that the application could be built and deployed using standard Java technologies without external dependencies. The study confirmed that socket programming, combined with Java Swing, could deliver a practical solution.

3. **System Design:**

   After confirming feasibility, system design activities began. The application was structured into two major components—Client and Server. Both components were built with modularity in mind, using Swing for UI and Sockets for communication. Message panels were created dynamically, with a consistent vertical layout and real-time updates.

4. **Data Flow and Process Modeling:**

   The data flow was designed to reflect bi-directional communication using sockets. Each message passes through a DataOutputStream and is received via DataInputStream. These are processed and rendered in real-time. Internally, a queue-like flow was created using a Box layout to align messages appropriately (left for received, right for sent).

5. **Identify System Requirements:**

   The system requires a Java Runtime Environment (JRE), basic hardware (2 GB RAM, 1 GHz CPU), and no database or external server. Network communication is established on localhost (127.0.0.1) using port 6001. GUI components such as JTextField, JPanel, JButton, and JLabel are used for the interface.

6. **Problem Identification and Solutions:**

   Problems identified include limited usability in console-based applications, lack of structured message rendering, and poor user experience. These issues were addressed by integrating a

responsive GUI with proper message formatting, timestamps, and structured layouts. Real-time updates were ensured by using continuous socket listeners on both sides.

7. **System Validation:**

   System validation was done by running both client and server modules simultaneously, exchanging messages, and confirming correct timestamp placement, alignment, and real-time rendering. Errors such as abrupt socket closure or message loss were tested and resolved during the validation stage.

8. **Risk Assessment:**

   Risks identified include socket timeout, GUI thread deadlock, and improper exception handling. These were mitigated using try-catch blocks, multithreading concepts, and proper stream closure methods to avoid crashes during runtime.

9. **Documentation:**

   Throughout the analysis phase, all functional requirements, technical constraints, and architectural decisions were documented. This includes inline comments, class-level explanations, and user instructions for running the project on any machine with Java installed.

10. **Project Timeline and Resource Allocation:**

    The project was developed over a short timeline of 1–2 weeks, with clear milestones such as GUI design, socket setup, message rendering, and final testing. As this is an individual academic project, resource allocation was limited to a single developer (you), using a single machine with Java and an IDE installed.

3.1 Existing System

The existing system of communication—such as command-line chat or email—lacks real-time interaction, visual formatting, and proper user engagement. Console-based Java chat systems have limited scope, no visual alignment, and no support for GUI-based interaction, making them difficult to use and unattractive for end-users.

Additionally, console chat systems do not display timestamps, which makes tracking message history unreliable. Errors are often unhandled, and sockets may disconnect without notification. There is no provision for modern UX elements like message bubbles, structured layouts, or closing the app via icons.

These limitations make it necessary to develop a desktop GUI application that can simulate a real-world chat environment using Java Swing and Networking.

3.2 Proposed System

The proposed Chatting Application is a Java-based desktop solution designed to enable real-time communication between two users through a client-server model. It uses **Java Swing** for building the Graphical User Interface and **Java Sockets** for managing network communication. The application provides a responsive and interactive interface where messages can be typed, sent, received, and displayed in a scrollable message panel.

Key features of the system include:

- Real-time bi-directional messaging using sockets (Socket, ServerSocket)
- Role-based structure (one user as Client, one as Server)
- GUI interface built with components like JPanel, JTextField, JButton, and JLabel
- Timestamped messages using SimpleDateFormat and Calendar
- Real-time rendering of messages with alignment (right for sent, left for received)
- Thread-safe socket communication and exception handling

This system replaces plain-text, non-interactive chat implementations with a modern, visually organized interface. It enhances the user experience, improves message readability, and serves as a strong example of combining GUI with backend networking in Java.

The proposed system is lightweight, easy to deploy, and suitable for use in academic, personal, or LAN-based messaging environments. It provides a foundation for expanding into group chat, media sharing, or even secure messaging in the future.

CHAPTER NO. 4

4. SYSTEM DESIGN

System Design for the Partner Management System defines the architecture, components, modules, data flow, and interfaces needed to build an efficient, scalable, and secure platform. It is divided into two main parts: **High-Level Design (HLD)** and **Low-Level Design (LLD)**.

. High-Level Design (HLD)

This outlines the overall structure of the system. It defines major components such as:

- **User Interface Layer:**
  Web-based front end for Admins, Partners, Employees, Users, and Shopkeepers. Each role accesses the platform through customized dashboards with role-specific functionalities.
- **Business Logic Layer:**
  Handles core operations like partner registration, offer approval, QR code generation, and notifications. It also manages workflows such as document validation, role routing, and status updates.
- **Data Access Layer:**
  Interacts with the SQL Server database to store and retrieve partner data, offers, user details, employee usage, and system logs.
- **Authentication Module:**
  Implements role-based access control, ensuring secure access to respective dashboards and functions based on user type (Admin, Partner, Employee, Shopkeeper, User).
- **Notification Module:**
  Sends real-time alerts (approval/rejection/status updates) via email or in-app messaging to users involved in different workflows.
- **Logging & Error Handling:**
  Uses tools like Serilog to log activities, track system performance, and capture exceptions for debugging and auditing purposes.

2. Low-Level Design (LLD)

This details each module's functionality, workflows, and interactions:

- **Partner Registration:**
  Form submission by partner including fields like name, email, and file upload for Aadhar and PAN. Data is validated and securely stored in the database with status tracking.
- **Offer Management:**
  Partners can submit new offers through their dashboard. Admins review each offer and either approve or reject them, providing reasoning and optional suggestions.
- **QR Code Generation:**
  Once an offer is approved, a unique QR code is generated and stored in the system. The code is linked to the offer and can be scanned for validation and redemption.
- **Dashboards:**
  Role-based interfaces are provided:
  - **Admin Dashboard:** Partner approvals, document viewing, offer validation
  - **Partner Dashboard:** Offer creation, status tracking
  - **User Dashboard:** View & validate offers
  - **Employee Dashboard:** View assigned offers
  - **Shopkeeper Dashboard:** Scan and validate QR codes
- **Status Tracking:**
  Offers go through statuses like *Pending*, *Approved*, *Rejected*, *Expired*, or *Used*. Partners and Admins can track changes in real-time.

Design Principles

- **Modular & Scalable:**
  System modules are separated logically to support ease of maintenance, independent scaling, and integration of future functionalities.
- **Secure:**
  Includes file validation, encrypted password storage, secure file handling for sensitive documents (Aadhar, PAN), and role-based permissions.
- **User-Friendly UI:**
  Developed using responsive, modern web technologies (HTML, CSS, JavaScript frameworks) ensuring intuitive navigation and accessibility.

4.1 Entity Relationship Diagram (ERD)

1. Admin

- **Attributes:**
  AdminID (PK), Name, Email, Password
- **Relationships:**
  ▪ Approves/Rejects → Partner
  ▪ Approves/Rejects → Offer

2. Partner

- **Attributes:**
  PartnerID (PK), Name, Email, Password, AadharNumber, PANNumber, Status
- **Relationships:**
  ▪ Submits → Offer

3. Offer

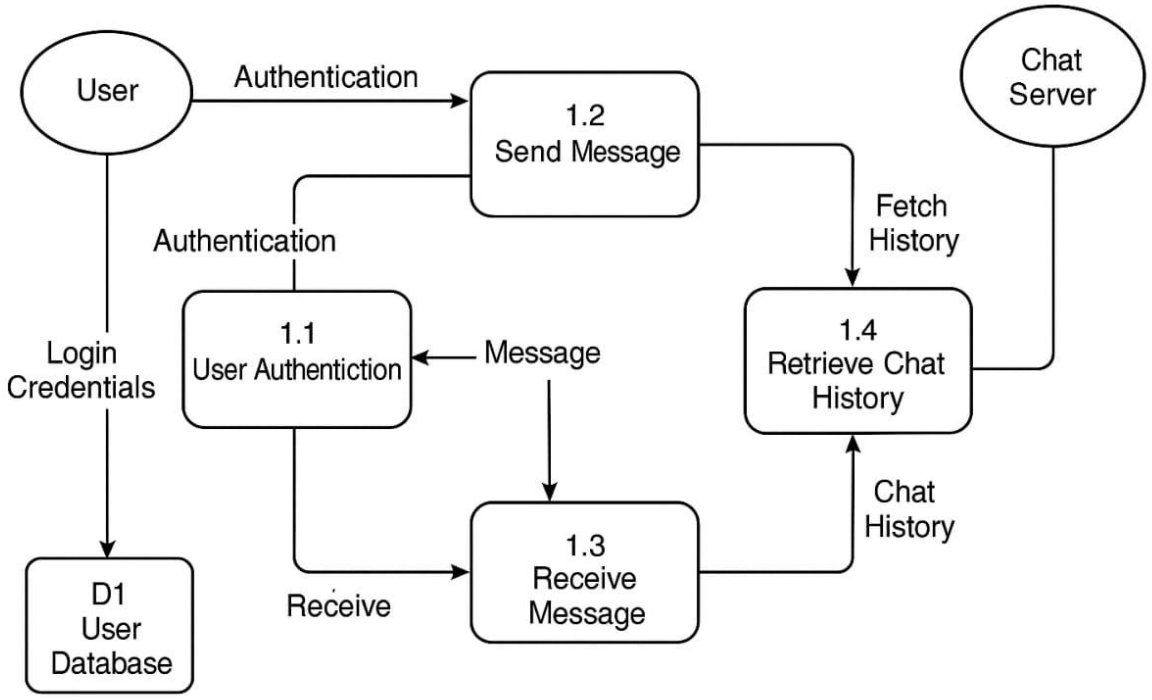
- **Attributes:**
  OfferID (PK), PartnerID (FK), OfferName, Description, StartDate, EndDate, Status, AdminMessage
- **Relationships:**
  ▪ Linked to → QRCode
  ▪ Visible to → Employee, User

4. Employee

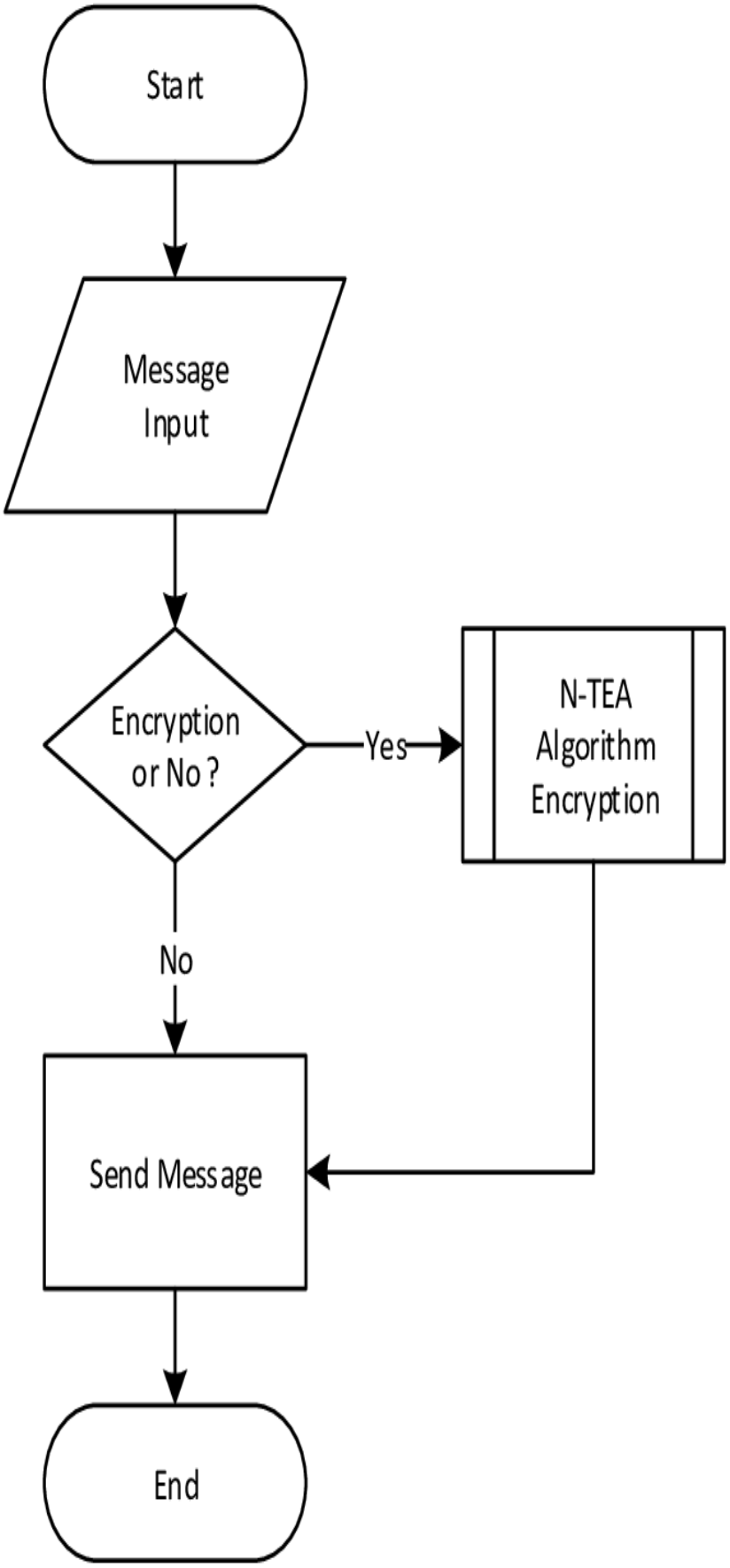
- **Attributes:**
  EmployeeID (PK), Name, Email, Password
- **Relationships:**
  ▪ Views/Uses → Offer

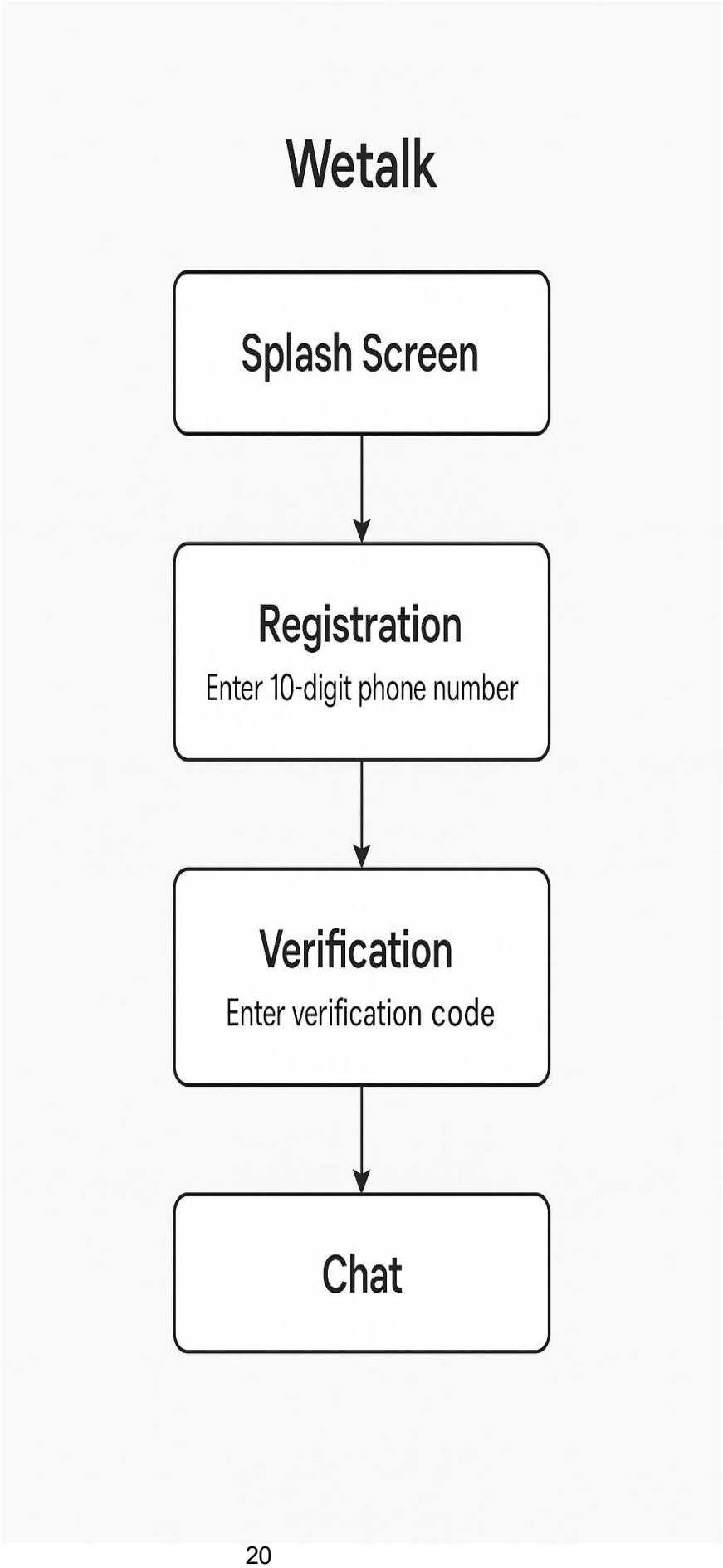
5. User

- **Attributes:**
  UserID (PK), Name, Email, Password
- **Relationships:**
  ▪ Views/Validates → Offer



ER Diagram for Chatting Application


Start
Message Input
Encryption or No ?
Yes
N-TEA Algorithm Encryption
No
Send Message
End


Wetalk
Splash Screen
Registration
Enter 10-digit phone number
Verification
Enter verification code
Chat

CHAPTER NO. 5

5. SYSTEM DEVELOPMENT

5.1Coding

5.1.1 Client.java

```
package chating.application;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.util.*;
import java.text.*;
import java.net.*;
import java.io.*;

public class Client implements ActionListener{
    JTextField text;
    JButton send;
    static JPanel a1;
    static Box vertical=Box.createVerticalBox();

    static JFrame f=new JFrame();

    static DataOutputStream dout;

    Client(){

        f.setLayout(null);

        JPanel p1=new JPanel();//JPanel p1=new JPanel(); //error aaye to aise likh dena
        p1.setBackground(new Color(7,94,84));
        p1.setBounds(0,0,450,70);//green color for backupper background
        p1.setLayout(null);
        f.add(p1);


      ImageIcon i1=new ImageIcon(ClassLoader.getSystemResource("icons/3.png"));
      Image i2=i1.getImage().getScaledInstance(25, 25, Image.SCALE_DEFAULT);
      ImageIcon i3=new ImageIcon(i2);
      JLabel back=new JLabel(i3);
      back.setBounds(5,20,25,25);
      p1.add(back);

      back.addMouseListener(new MouseAdapter(){
         //@Override mouseclicked event
         public void mouseClicked(MouseEvent ae){
            System.exit(0);
         }
      });

      ImageIcon i4=new ImageIcon(ClassLoader.getSystemResource("icons/pp.jpg"));
```

```
Image i5=i4.getImage().getScaledInstance(50, 50,Image.SCALE_DEFAULT);
ImageIcon i6=new ImageIcon(i5);
JLabel profile=new JLabel(i6);
profile.setBounds(40,10,50,50);
p1.add(profile);

ImageIcon i7=new ImageIcon(ClassLoader.getSystemResource("icons/video.png"));
Image i8=i7.getImage().getScaledInstance(30, 30,Image.SCALE_DEFAULT);
ImageIcon i9=new ImageIcon(i8);
JLabel video=new JLabel(i9);
video.setBounds(300,20,30,30);
p1.add(video);

ImageIcon i10=new ImageIcon(ClassLoader.getSystemResource("icons/phone.png"));
Image i11=i10.getImage().getScaledInstance(35, 30,Image.SCALE_DEFAULT);
ImageIcon i12=new ImageIcon(i11);
JLabel phone=new JLabel(i12);
phone.setBounds(360,20,35,30);
p1.add(phone);

ImageIcon i13=new ImageIcon(ClassLoader.getSystemResource("icons/3icon.png"));
Image i14=i13.getImage().getScaledInstance(10, 25,Image.SCALE_DEFAULT);
ImageIcon i15=new ImageIcon(i14);
JLabel morevert=new JLabel(i15);
morevert.setBounds(420,20,10,25);
p1.add(morevert);

JLabel name=new JLabel("CLIENT");
name.setBounds(110,15,100,18);
name.setForeground(Color.WHITE);
name.setFont(new Font("SAN_SERIF",Font.BOLD,18));
p1.add(name);

JLabel status=new JLabel("Online");
status.setBounds(110,35,100,18);
status.setForeground(Color.WHITE);
status.setFont(new Font("SAN_SERIF",Font.BOLD,18));
p1.add(status);

  a1=new JPanel();
  a1.setBounds(5,75,440,570);
  f.add(a1);

text=new JTextField();
text.setBounds(5,655,310,40);
//text.setFont(new Font("SAN_SERIF",Font.BOLD,14"));
text.setFont(new Font("SAN_SERIF",Font.PLAIN,16));
f.add(text);

send=new JButton("Send");
send.setBounds(320,655,123,40);
send.setBackground(new Color(7,94,84));
send.setForeground(Color.WHITE);
send.addActionListener(this);
send.setFont(new Font("SAN_SERIF",Font.PLAIN,16));
```

```
  f.add(send);
  f.setSize(450,700);
  f.setLocation(800,20);
  f.setUndecorated(true);
  f.getContentPane().setBackground(Color.white);
  f.setVisible(true);
}
public void actionPerformed(ActionEvent ae) {
     try{
   String out=text.getText();
   JPanel p2=formatLabel(out);
   a1.setLayout(new BorderLayout());
   JPanel right=new JPanel(new BorderLayout());
   right.add(p2,BorderLayout.LINE_END);
   vertical.add(right);
   vertical.add(Box.createVerticalStrut(15));
   a1.add(vertical,BorderLayout.PAGE_START);
   dout.writeUTF(out);
   text.setText("");
   f.repaint();
   f.invalidate();
   f.validate();
     }catch(Exception e){
      e.printStackTrace();
     }
 }
 public static JPanel formatLabel(String out){
   JPanel panel=new JPanel();
   panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));

   JLabel output=new JLabel("<html><p> <style=\"width:150px\">" + out +"</p></html>");
   output.setFont(new Font("Tahoma",Font.PLAIN,18));
   output.setBackground(new Color(37,211,102));
   output.setOpaque(true);
   output.setBorder(new EmptyBorder(15,15,15,50));

   panel.add(output);
   Calendar cal=Calendar.getInstance();
   SimpleDateFormat sdf=new SimpleDateFormat("hh:mm");
   JLabel time=new JLabel();
   time.setText(sdf.format(cal.getTime()));
   panel.add(time);
   return panel;
 }
public static  void main(String[] args){
    new Client();
   try{
      Socket s=new Socket("127.0.0.1",6001);
      DataInputStream din=new DataInputStream(s.getInputStream());
      dout=new DataOutputStream(s.getOutputStream());
      while(true){
          a1.setLayout(new BorderLayout());
        String msg=din.readUTF();
        JPanel panel=formatLabel(msg);
        JPanel left=new JPanel(new BorderLayout());
```

```
            left.add(panel,BorderLayout.LINE_START);
            vertical.add(left);
            vertical.add(Box.createVerticalStrut(15));
            a1.add(vertical,BorderLayout.PAGE_START);
            f.validate();
          }
        } catch(Exception e){
          e.printStackTrace();
        }
    }

}
```

5.1.2 Server.java

```
package chating.application;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.util.*;
import java.text.*;
import java.net.*;
import java.io.*;

public class Server implements ActionListener{
   JTextField text;
   JButton send;
   JPanel a1;
   static Box vertical=Box.createVerticalBox();
   static JFrame f=new JFrame();
   static DataOutputStream dout;
   Server(){

      f.setLayout(null);

      JPanel p1=new JPanel();//JPanel p1=new JPanel(); //error aaye to aise
likh dena
      p1.setBackground(new Color(7,94,84));
      p1.setBounds(0,0,450,70);//green color for backupper background
      p1.setLayout(null);
      f.add(p1);

     ImageIcon i1=new
ImageIcon(ClassLoader.getSystemResource("icons/3.png"));
     Image i2=i1.getImage().getScaledInstance(25, 25,
Image.SCALE_DEFAULT);
     ImageIcon i3=new ImageIcon(i2);
     JLabel back=new JLabel(i3);
     back.setBounds(5,20,25,25);
     p1.add(back);

     back.addMouseListener(new MouseAdapter(){
        //@Override
        public void mouseClicked(MouseEvent ae){
           System.exit(0);
        }
     });

     ImageIcon i4=new
ImageIcon(ClassLoader.getSystemResource("icons/r.png"));
     Image i5=i4.getImage().getScaledInstance(50,
50,Image.SCALE_DEFAULT);
     ImageIcon i6=new ImageIcon(i5);
     JLabel profile=new JLabel(i6);
     profile.setBounds(40,10,50,50);
```

```
    p1.add(profile);

    ImageIcon i7=new
ImageIcon(ClassLoader.getSystemResource("icons/video.png"));
    Image i8=i7.getImage().getScaledInstance(30,
30,Image.SCALE_DEFAULT);
    ImageIcon i9=new ImageIcon(i8);
    JLabel video=new JLabel(i9);
    video.setBounds(300,20,30,30);
    p1.add(video);

    ImageIcon i10=new
ImageIcon(ClassLoader.getSystemResource("icons/phone.png"));
    Image i11=i10.getImage().getScaledInstance(35,
30,Image.SCALE_DEFAULT);
    ImageIcon i12=new ImageIcon(i11);
    JLabel phone=new JLabel(i12);
    phone.setBounds(360,20,35,30);
    p1.add(phone);

    ImageIcon i13=new
ImageIcon(ClassLoader.getSystemResource("icons/3icon.png"));
    Image i14=i13.getImage().getScaledInstance(10,
25,Image.SCALE_DEFAULT);
    ImageIcon i15=new ImageIcon(i14);
    JLabel morevert=new JLabel(i15);
    morevert.setBounds(420,20,10,25);
    p1.add(morevert);

    JLabel name=new JLabel("BOAT");
    name.setBounds(110,15,100,18);
    name.setForeground(Color.WHITE);
    name.setFont(new Font("SAN_SERIF",Font.BOLD,18));
    p1.add(name);

    JLabel status=new JLabel("Online");
    status.setBounds(110,35,100,18);
    status.setForeground(Color.WHITE);
    status.setFont(new Font("SAN_SERIF",Font.BOLD,18));
    p1.add(status);

     a1=new JPanel();
     a1.setBounds(5,75,440,570);
     f.add(a1);

   text=new JTextField();
   text.setBounds(5,655,310,40);
   //text.setFont(new Font("SAN_SERIF",Font.BOLD,14"));
   text.setFont(new Font("SAN_SERIF",Font.PLAIN,16));
   f.add(text);

   send=new JButton("Send");
   send.setBounds(320,655,123,40);
   send.setBackground(new Color(7,94,84));
   send.setForeground(Color.WHITE);
```

```
    send.addActionListener(this);
    send.setFont(new Font("SAN_SERIF",Font.PLAIN,16));
    f.add(send);

     f.setSize(450,700);
     f.setLocation(200,20);
     f.setUndecorated(true);
     f.getContentPane().setBackground(Color.white);
     f.setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
       try{
      String out=text.getText();
      JPanel p2=formatLabel(out); //p2.add(output);
      a1.setLayout(new BorderLayout());
      JPanel right=new JPanel(new BorderLayout());
      right.add(p2,BorderLayout.LINE_END);
      vertical.add(right);
      vertical.add(Box.createVerticalStrut(15));
      a1.add(vertical,BorderLayout.PAGE_START);
      dout.writeUTF(out);
      text.setText("");
      f.repaint();
      f.invalidate();
      f.validate();
      }  catch(Exception e){
         e.printStackTrace();
      }
    }
    public static JPanel formatLabel(String out){
      JPanel panel=new JPanel();
      panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));

      JLabel output=new JLabel("<html><p> <style=\"width:150px\">" + out
+"</p></html>");
      output.setFont(new Font("Tahoma",Font.PLAIN,18));
      output.setBackground(new Color(37,211,102));
      output.setOpaque(true);
      output.setBorder(new EmptyBorder(15,15,15,50));

      panel.add(output);
      Calendar cal=Calendar.getInstance();
      SimpleDateFormat sdf=new SimpleDateFormat("hh:mm");
      JLabel time=new JLabel();
      time.setText(sdf.format(cal.getTime()));
      panel.add(time);
      return panel;
    }
   public static  void main(String[] args){
       new Server();

       try{
          ServerSocket skt=new ServerSocket(6001);
          while(true){
```

```
                Socket s=skt.accept();
                DataInputStream din=new DataInputStream(s.getInputStream());
                dout=new DataOutputStream(s.getOutputStream());

                while(true){
                   String msg=din.readUTF();
                   JPanel panel=formatLabel(msg);

                   JPanel left=new JPanel(new BorderLayout());
                   left.add(panel,BorderLayout.LINE_START);
                   vertical.add(left);
                   f.validate();
                }

              }
            }catch(Exception e){
               e.printStackTrace();
            }


        }

}
```

5.1.3 RegisterPage.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Random;

public class RegisterPage extends JFrame {
    JTextField phoneField, emailField;
    JButton getOtpBtn, nextBtn;
    JLabel phoneOtpLabel, emailOtpLabel;
    String phoneOtp = "";
    String emailOtp = "";

    public RegisterPage() {
        setTitle("UserA Registration");
        setSize(300, 400);
        setLayout(new GridLayout(8, 1, 5, 5));
        setLocation(300, 200);

        phoneField = new JTextField();
        emailField = new JTextField();
        getOtpBtn = new JButton("Get OTP");
        nextBtn = new JButton("Next");

        phoneOtpLabel = new JLabel();
        emailOtpLabel = new JLabel();

        add(new JLabel("Phone Number:"));
        add(phoneField);
        add(new JLabel("Email ID:"));
        add(emailField);
        add(getOtpBtn);
        add(phoneOtpLabel);
        add(emailOtpLabel);
        add(nextBtn);

        getOtpBtn.setBackground(new Color(7, 94, 84));
        getOtpBtn.setForeground(Color.WHITE);
        nextBtn.setBackground(new Color(7, 94, 84));
        nextBtn.setForeground(Color.WHITE);

        getOtpBtn.addActionListener(e -> generateAndShowOtp());

        nextBtn.addActionListener(e -> {
            if (!phoneOtp.isEmpty() && !emailOtp.isEmpty()) {
                new NamePage();
                dispose();
            } else {
                JOptionPane.showMessageDialog(this, "Please generate OTP
first.");
            }
        });
```

```
        setVisible(true);
    }

    void generateAndShowOtp() {
        phoneOtp = generateOtp();
        emailOtp = generateOtp();

        phoneOtpLabel.setText("Phone OTP: " + phoneOtp);
        emailOtpLabel.setText("Email OTP: " + emailOtp);
    }

    String generateOtp() {
        Random rand = new Random();
        int otp = 1000 + rand.nextInt(9000);
        return String.valueOf(otp);
    }

    public static void main(String[] args) {
        new RegisterPage();
    }
}
```

5.1.3 LoginPage.java

```
package ChatUI;
import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import javax.swing.*;

public class LoginPage extends JFrame {
    JTextField phoneField, emailField;
    JButton getOtpBtn, loginBtn;
    JLabel phoneOtpLabel, emailOtpLabel;
    String phoneOtp = "";
    String emailOtp = "";

    public LoginPage() {
        setTitle("UserA Login");
        setSize(300, 400);
        setLocation(300, 200);
        setLayout(new GridLayout(8, 1, 5, 5));

        phoneField = new JTextField();
        emailField = new JTextField();
        getOtpBtn = new JButton("Get OTP");
        loginBtn = new JButton("Login");

        phoneOtpLabel = new JLabel();
        emailOtpLabel = new JLabel();

        add(new JLabel("Phone Number:"));
        add(phoneField);
        add(new JLabel("Email ID:"));
        add(emailField);
        add(getOtpBtn);
        add(phoneOtpLabel);
        add(emailOtpLabel);
        add(loginBtn);

        getOtpBtn.setBackground(new Color(7, 94, 84));
        getOtpBtn.setForeground(Color.WHITE);
        loginBtn.setBackground(new Color(7, 94, 84));
        loginBtn.setForeground(Color.WHITE);

        getOtpBtn.addActionListener(e -> generateAndShowOtp());

        loginBtn.addActionListener(e -> verifyAndLogin());

        emailField.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent ke) {
                if (ke.getKeyCode() == KeyEvent.VK_ENTER) {
                    verifyAndLogin();
                }
            }
        });
```

```
        setVisible(true);
    }

    void generateAndShowOtp() {
        phoneOtp = generateOtp();
        emailOtp = generateOtp();

        phoneOtpLabel.setText("Phone OTP: " + phoneOtp);
        emailOtpLabel.setText("Email OTP: " + emailOtp);
    }

    String generateOtp() {
        Random rand = new Random();
        int otp = 1000 + rand.nextInt(9000);
        return String.valueOf(otp);
    }

    void verifyAndLogin() {
        if (!phoneOtp.isEmpty() && !emailOtp.isEmpty()) {
            new NamePage();
            dispose();
        } else {
            JOptionPane.showMessageDialog(this, "Please generate OTP first.");
        }
    }

    public static void main(String[] args) {
        new LoginPage();
    }
}
```

5.1.4 NamePage.java

```
package ChatUI;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class NamePage extends JFrame {
   JTextField nameField;
   JButton readyBtn;

   public NamePage() {
      setTitle("Enter Name");
      setSize(300, 200);
      setLayout(new BorderLayout());

      nameField = new JTextField();
      readyBtn = new JButton("Ready to Chat");

      add(new JLabel("Enter your name:"), BorderLayout.NORTH);
      add(nameField, BorderLayout.CENTER);
      add(readyBtn, BorderLayout.SOUTH);

      readyBtn.addActionListener(e -> {
         String name = nameField.getText().trim();
         if (!name.isEmpty()) {
            UserData.displayName = name;
            new UserA();
            dispose();
         } else {
            JOptionPane.showMessageDialog(this, "Please enter your
name");
         }
      });

      setVisible(true);
      setLocation(300, 200);
   }
}
```

5.1.5 UserA.java

```
package ChatUI;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class UserA extends JFrame {

    JPanel chatPanel;
    JTextField inputField;
    JButton sendButton;
    static Box vertical = Box.createVerticalBox();
    static DataOutputStream dout;
    static String displayName = UserData.displayName;

    UserA() {
        setTitle("Chat - " + displayName);
        setSize(400, 600);
        setLocation(200, 100);
        setLayout(new BorderLayout());

        JPanel topPanel = new JPanel();
        topPanel.setBackground(new Color(7, 94, 84));
        topPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        JLabel nameLabel = new JLabel(displayName);
        nameLabel.setForeground(Color.WHITE);
        nameLabel.setFont(new Font("SAN_SERIF", Font.BOLD, 18));
        topPanel.add(nameLabel);
        add(topPanel, BorderLayout.NORTH);

        chatPanel = new JPanel();
        chatPanel.setLayout(new BoxLayout(chatPanel, BoxLayout.Y_AXIS));

        JScrollPane scrollPane = new JScrollPane(chatPanel);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROL
LBAR_ALWAYS);
        add(scrollPane, BorderLayout.CENTER);

        JPanel inputPanel = new JPanel(new BorderLayout());
        inputField = new JTextField();
        sendButton = new JButton("Send");
        sendButton.setBackground(new Color(7, 94, 84));
        sendButton.setForeground(Color.WHITE);

        inputPanel.add(inputField, BorderLayout.CENTER);
        inputPanel.add(sendButton, BorderLayout.EAST);
        add(inputPanel, BorderLayout.SOUTH);
```

```
        sendButton.addActionListener(e -> sendMessage());
        inputField.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent ke) {
                if (ke.getKeyCode() == KeyEvent.VK_ENTER) {
                    sendMessage();
                }
            }
        });

        setVisible(true);

        try {
            ServerSocket serverSocket = new ServerSocket(6001);
            Socket socket = serverSocket.accept();

            DataInputStream din = new
DataInputStream(socket.getInputStream());
            dout = new DataOutputStream(socket.getOutputStream());

            while (true) {
                String msg = din.readUTF();
                addReceivedMessage(msg);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    void sendMessage() {
        try {
            String msg = inputField.getText().trim();
            if (!msg.isEmpty()) {
                addSentMessage(msg);
                dout.writeUTF(msg);
                inputField.setText("");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    void addSentMessage(String msg) {
        JPanel messagePanel = formatLabel(msg, true);
        chatPanel.add(messagePanel);
        vertical.add(Box.createVerticalStrut(10));
        chatPanel.add(vertical);
        chatPanel.revalidate();
        chatPanel.repaint();
    }

    void addReceivedMessage(String msg) {
        JPanel messagePanel = formatLabel(msg, false);
        chatPanel.add(messagePanel);
        vertical.add(Box.createVerticalStrut(10));
```

```
        chatPanel.add(vertical);
        chatPanel.revalidate();
        chatPanel.repaint();
    }

    JPanel formatLabel(String msg, boolean sentByUserA) {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        JLabel messageLabel = new JLabel("<html><p style='width:150px'>" +
msg + "</p></html>");
        messageLabel.setFont(new Font("Tahoma", Font.PLAIN, 16));
        messageLabel.setBackground(sentByUserA ? new Color(37, 211, 102) :
new Color(220, 248, 198));
        messageLabel.setOpaque(true);
        messageLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
        String time = sdf.format(new Date());
        JLabel timeLabel = new JLabel(time);
        timeLabel.setFont(new Font("Tahoma", Font.PLAIN, 10));

        if (sentByUserA) {
            panel.add(messageLabel);
            panel.add(timeLabel);
            panel.setAlignmentX(Component.RIGHT_ALIGNMENT);
        } else {
            panel.add(messageLabel);
            panel.add(timeLabel);
            panel.setAlignmentX(Component.LEFT_ALIGNMENT);
        }

        return panel;
    }

    public static void main(String[] args) {
        new UserA();
    }
}
```

5.1.6 UserB.java

```
package ChatUI;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
```

```
import java.util.Date;

public class UserB extends JFrame {

    JPanel chatPanel;
    JTextField inputField;
    JButton sendButton;
    static Box vertical = Box.createVerticalBox();
    static DataOutputStream dout;
    static String displayName = UserDataB.displayName;

    UserB() {
        setTitle("Chat - " + displayName);
        setSize(400, 600);
        setLocation(700, 100);
        setLayout(new BorderLayout());

        JPanel topPanel = new JPanel();
        topPanel.setBackground(new Color(7, 94, 84));
        topPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        JLabel nameLabel = new JLabel(displayName);
        nameLabel.setForeground(Color.WHITE);
        nameLabel.setFont(new Font("SAN_SERIF", Font.BOLD, 18));
        topPanel.add(nameLabel);
        add(topPanel, BorderLayout.NORTH);

        chatPanel = new JPanel();
        chatPanel.setLayout(new BoxLayout(chatPanel, BoxLayout.Y_AXIS));

        JScrollPane scrollPane = new JScrollPane(chatPanel);
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROL
LBAR_ALWAYS);
        add(scrollPane, BorderLayout.CENTER);

        JPanel inputPanel = new JPanel(new BorderLayout());
        inputField = new JTextField();
        sendButton = new JButton("Send");
        sendButton.setBackground(new Color(7, 94, 84));
        sendButton.setForeground(Color.WHITE);

        inputPanel.add(inputField, BorderLayout.CENTER);
        inputPanel.add(sendButton, BorderLayout.EAST);
        add(inputPanel, BorderLayout.SOUTH);

        sendButton.addActionListener(e -> sendMessage());
        inputField.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent ke) {
                if (ke.getKeyCode() == KeyEvent.VK_ENTER) {
                    sendMessage();
                }
            }
        });

        setVisible(true);
```

```
        try {
            Socket socket = new Socket("127.0.0.1", 6001);

            DataInputStream din = new
DataInputStream(socket.getInputStream());
            dout = new DataOutputStream(socket.getOutputStream());

            while (true) {
                String msg = din.readUTF();
                addReceivedMessage(msg);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    void sendMessage() {
        try {
            String msg = inputField.getText().trim();
            if (!msg.isEmpty()) {
                addSentMessage(msg);
                dout.writeUTF(msg);
                inputField.setText("");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    void addSentMessage(String msg) {
        JPanel messagePanel = formatLabel(msg, true);
        chatPanel.add(messagePanel);
        vertical.add(Box.createVerticalStrut(10));
        chatPanel.add(vertical);
        chatPanel.revalidate();
        chatPanel.repaint();
    }

    void addReceivedMessage(String msg) {
        JPanel messagePanel = formatLabel(msg, false);
        chatPanel.add(messagePanel);
        vertical.add(Box.createVerticalStrut(10));
        chatPanel.add(vertical);
        chatPanel.revalidate();
        chatPanel.repaint();
    }

    JPanel formatLabel(String msg, boolean sentByUserB) {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        JLabel messageLabel = new JLabel("<html><p style='width:150px'>" +
msg + "</p></html>");
        messageLabel.setFont(new Font("Tahoma", Font.PLAIN, 16));
```

```
        messageLabel.setBackground(sentByUserB ? new Color(37, 211, 102) :
new Color(220, 248, 198));
        messageLabel.setOpaque(true);
        messageLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
        String time = sdf.format(new Date());
        JLabel timeLabel = new JLabel(time);
        timeLabel.setFont(new Font("Tahoma", Font.PLAIN, 10));

        if (sentByUserB) {
            panel.add(messageLabel);
            panel.add(timeLabel);
            panel.setAlignmentX(Component.RIGHT_ALIGNMENT);
        } else {
            panel.add(messageLabel);
            panel.add(timeLabel);
            panel.setAlignmentX(Component.LEFT_ALIGNMENT);
        }

        return panel;
    }

    public static void main(String[] args) {
        new RegisterPageB();
    }
}
```

CHAPTER NO. 6

6. TESTING

6.1 Testing

Testing is a critical phase in the software development lifecycle, aimed at ensuring the correctness, reliability, and performance of the Partner Management System. Throughout development, various levels of testing were conducted, including unit testing, integration testing, system testing, and user acceptance testing (UAT). Unit testing focused on verifying individual modules like authentication, offer creation, and QR validation to ensure each component behaved as expected. Integration testing was used to confirm that these components worked together seamlessly, particularly in workflows involving role-based actions such as partner registration and admin approvals.

System testing evaluated the entire application for performance, usability, and compatibility across browsers and devices. Test cases were designed to validate each functional requirement, such as proper redirection based on user roles, file upload validations, session timeouts, and error handling. Manual and automated tests were executed to detect and fix bugs. Verification ensured the system met design specifications, while validation confirmed it fulfilled user requirements. Additionally, regression testing was conducted after updates to make sure new changes didn't affect existing features.

Debugging tools and detailed logging (via Serilog) helped identify and resolve issues efficiently. The testing phase concluded with user feedback, which played a key role in fine-tuning the final deployment, ensuring that the system performs reliably under real-world conditions.

6.2 Testing Cases

Several functional and non-functional testing cases were designed to validate the core features of the Partner Management System. Each test case was structured to cover specific functionalities like user login, partner registration, offer creation, file uploads, admin approval/rejection workflows, QR code generation and validation, and secure logout.

For instance, one test case checked whether a partner could successfully upload Aadhaar and PAN documents and receive confirmation, while another tested if the admin could approve or reject the offer and send feedback. Boundary value tests were used for validating input fields, such as username length or file size limits. Error-handling test cases ensured that invalid logins, expired sessions, and missing data were handled gracefully with appropriate messages. Test scenarios were also created to simulate multiple user roles accessing the system concurrently, ensuring session handling and authorization controls functioned correctly. Moreover, the notification system (email or in-app) was tested for timely delivery and content accuracy.

Each test case included the test scenario, input data, expected result, and actual result, which helped in identifying and fixing issues during debugging. These cases ensured that all features worked as intended and the user experience remained consistent across different use cases.

Detailed test cases were designed and executed for each module of the system. For example:

- **Login Module Test Cases:**
  - Valid and invalid login credentials
  - Role-based access (e.g., Admin cannot access Partner panel)
  - Session timeout handling
- **Partner Registration Test Cases:**
  - Uploading valid/invalid Aadhaar and PAN documents
  - Duplicate email detection
  - Field validation (e.g., mandatory fields)
- **Offer Module Test Cases:**
  - Creating an offer with valid/invalid date ranges
  - Offer approval/rejection workflow
  - Notification triggers on status change

6.3 Verification and Validation

The system underwent both verification and validation to ensure it met specifications and real-world

expectations.

- **Verification**: involved static and dynamic testing methods. Code reviews, requirement traceability checks, and walkthroughs were conducted to ensure that the development aligned with documented specifications. Each module was verified to ensure correct implementation logic.
- **Validation**: confirmed that the system solved real problems faced by stakeholders. Actual users, including admin personnel, shopkeepers, partners, and employees, interacted with the system in simulated production environments. Feedback was collected to make iterative improvements. Real-world scenarios such as expired offers, late approvals, invalid document uploads, and simultaneous logins were tested to validate performance, accuracy, and user satisfaction.

Verification was conducted throughout the development cycle to ensure that each module was built according to the technical specifications and system design. This included verifying core functionalities such as partner registration with document uploads (Aadhaar and PAN), role-based authentication (Admin and Partner), offer creation and submission by partners, and offer approval/rejection by the admin. Code reviews, requirement traceability matrices, and module testing were performed to verify that these components aligned with the system's architecture and logic.

Validation, on the other hand, was done to confirm that the system fulfilled end-user needs and business goals. Validation involved executing real-world test scenarios, such as a partner registering and submitting an offer, followed by the admin reviewing and responding with approval or rejection, which then triggered an email or in-app notification. Another example included testing whether employees could view only admin-approved offers, validate them to generate a unique QR code, and whether that QR could be scanned by shopkeepers to mark offers as used.

6.3 Debugging

Debugging was made efficient through the use of logging tools like Serilog, which captured system events, user actions, and exceptions in a centralized log file. This made it easier to trace issues like incorrect file formats, login failures, or offer mismatches. Error messages were designed to be informative for developers while user-facing alerts were kept simple and friendly. Debugging tools integrated with Visual Studio (e.g., breakpoints, watch windows, and stack traces) helped identify logical flaws and runtime errors. The use of layered architecture (controller, service, repository) also improved traceability and made isolating issues quicker during testing and debugging.

Debugging is a crucial process in software development that involves identifying, isolating, and fixing errors or unexpected behavior in the code. During the development of the Partner Management System, debugging was extensively used to ensure that the application operated smoothly across all modules, including partner registration, file uploads, role-based authentication, offer approval, QR code generation, and dashboard navigation.

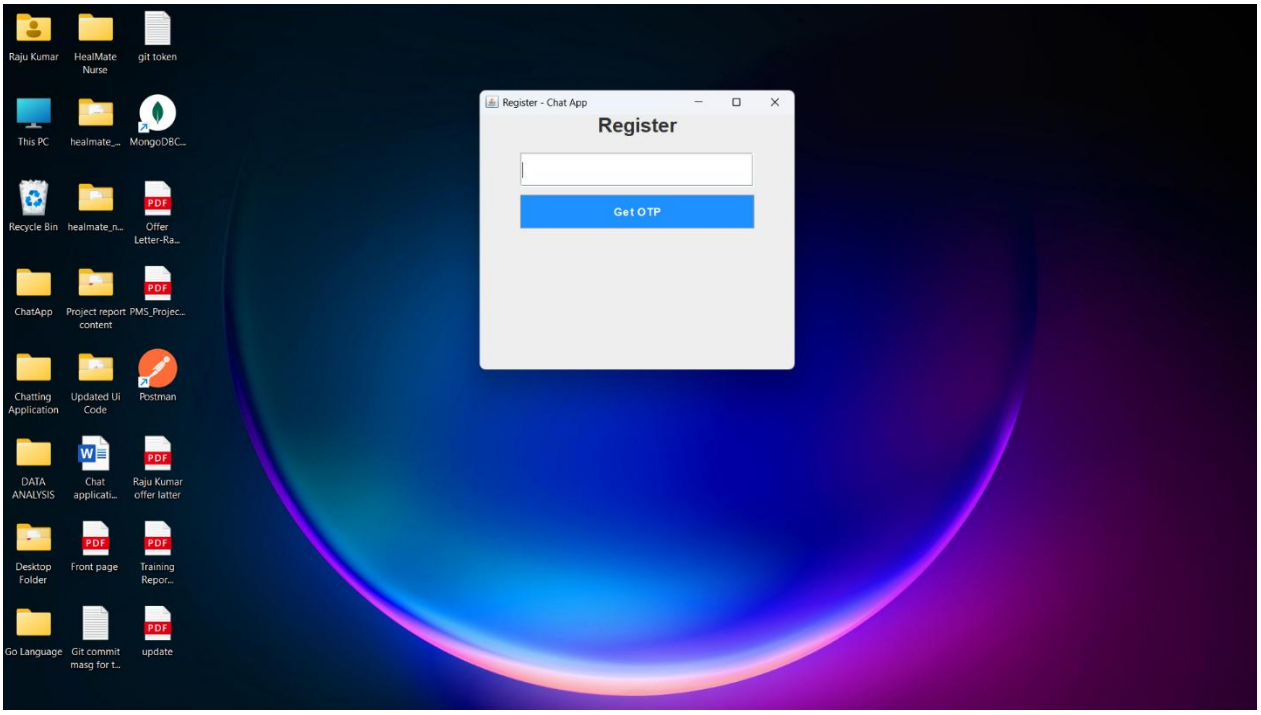
Common issues encountered included incorrect redirects based on user roles, missing validation messages during document uploads, and offer approval logic failures. These were addressed using tools such as Visual Studio's built-in debugger, which allowed step-by-step code tracing, and browser developer tools for front-end debugging. Real-time logs were also captured using Serilog, which provided detailed insights into runtime errors, failed API calls, and database issues, significantly reducing troubleshooting time.

For example, if a partner's uploaded PAN card file failed to save in the database, logs helped identify whether the issue was due to file size, format mismatch, or server-side handling. Similarly, if a shopkeeper was unable to scan a QR code, debugging focused on validating the QR payload, decoding logic, and ensuring proper status update post-scan.

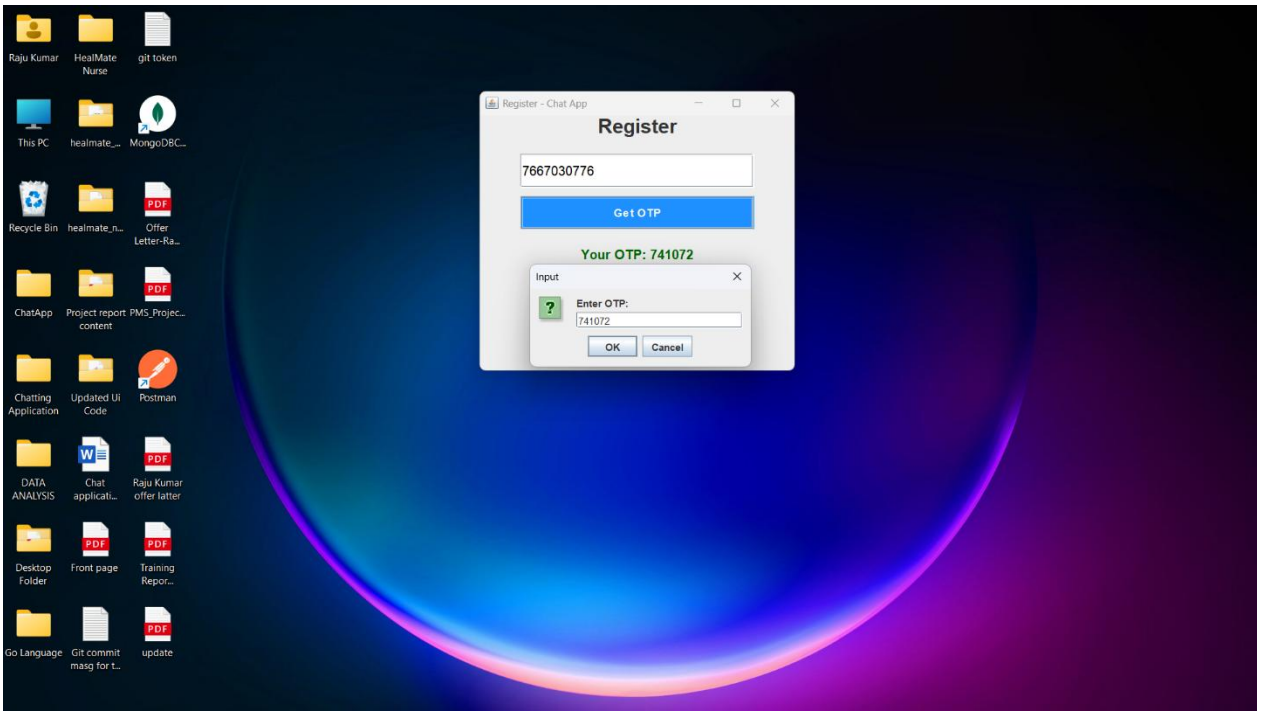
Overall, debugging ensured stability, accuracy, and reliability of the system, helping the development team deliver a robust and error-free application.

CHAPTER NO. 7

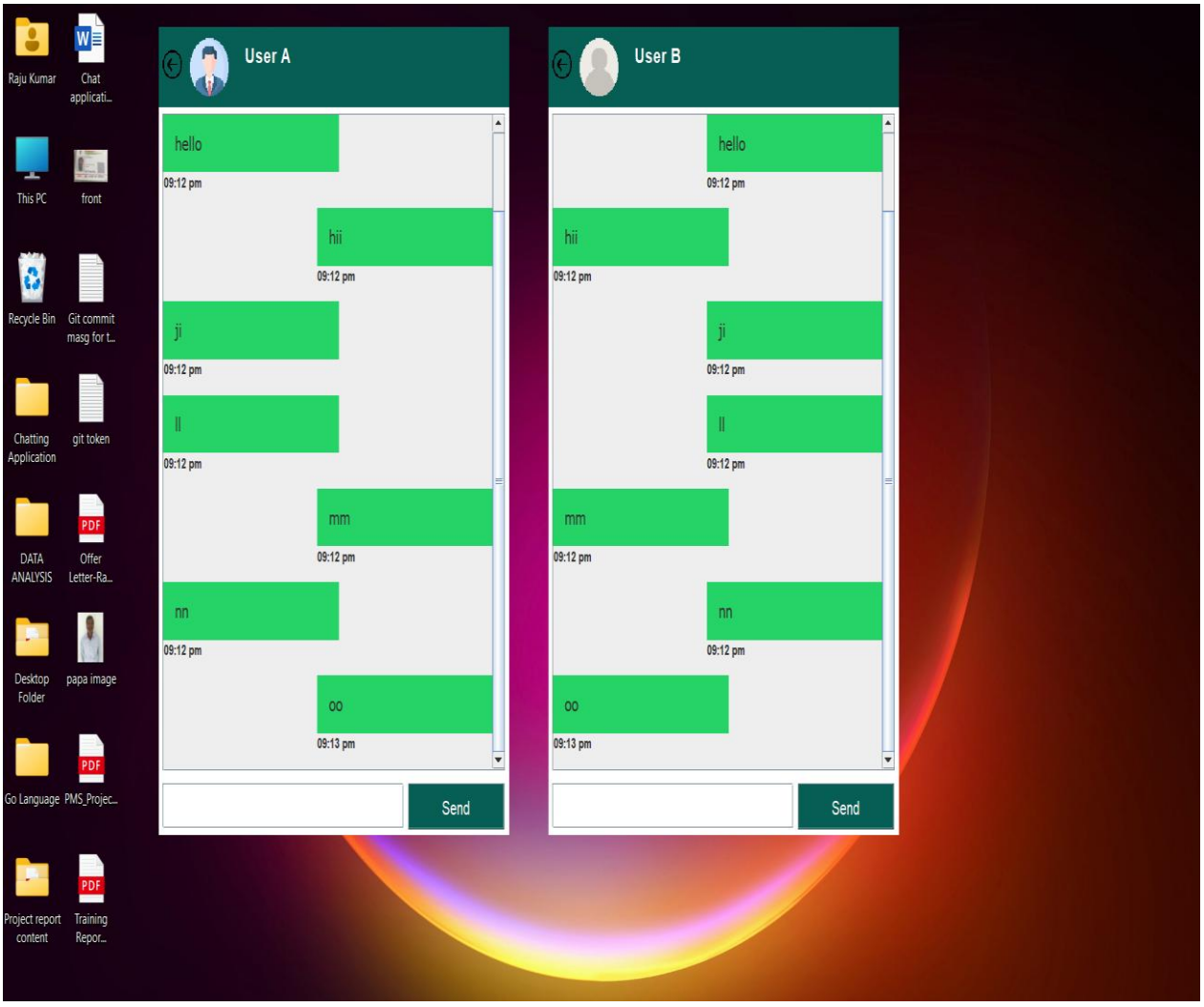
7.SCREENSHOTS



(Fig. 7.1) Register Page



(Fig. 7.2) Login & Signup Page



(Fig. 7.3) Dashboard Page

CHAPTER NO. 8

8.1 IMPLEMENTATION

Security plays a crucial role in the development and deployment of the Partner Management System, as it deals with sensitive user data, identity documents (such as Aadhar and PAN), and access controls for different roles like Admin, Partner, Employee, Shopkeeper, and User. To ensure a secure environment, various security mechanisms were implemented throughout the application. The system uses strong authentication and authorization techniques using **ASP.NET Core Identity**, ensuring that only verified users can access specific parts of the application based on their roles. **Role-based access control (RBAC)** is enforced to segregate permissions for different user types, restricting unauthorized access to admin panels, offer workflows, and document views. Passwords are securely hashed using **built-in cryptographic hashing algorithms** before being stored in the database, protecting against unauthorized access even in case of a data breach.

To enhance **data security**, the system enforces **HTTPS protocols**, ensuring encrypted communication between the client and server, preventing man-in-the-middle (MITM) attacks. Additionally, **critical user information and identity documents** are encrypted both in transit and at rest. File uploads, particularly Aadhar and PAN documents, undergo **strict validation checks** on format (e.g., PDF, JPEG), size restrictions, and MIME type validation. Uploaded documents are stored in **protected server directories**, accessible only to the Admin via secure API endpoints. **Input validation** is rigorously implemented at both the client and server sides to prevent security vulnerabilities such as **SQL injection**, **cross-site scripting (XSS)**, and **cross-site request forgery (CSRF)**. ASP.NET Core's **built-in anti-forgery token mechanism** is utilized for protecting form submissions. Furthermore, the use of **Entity Framework Core** with parameterized queries ensures secure interaction with the database, preventing SQL-based attacks.

To maintain accountability and enable traceability, **Serilog** is integrated as a structured logging tool. It logs critical events such as login attempts, failed authentications, offer submissions, file uploads, QR code validations, and admin decisions. These logs not only help during debugging and audits but also act as a key element in detecting suspicious behavior or anomalies in user activity.

Session management is handled carefully with secure cookies, session timeouts, and token expiration policies to ensure that inactive users are logged out automatically, reducing the risk of session hijacking. A secure logout feature is implemented to clear user sessions and authentication tokens from both client and server.

The **frontend** dynamically adjusts UI elements, navigation menus, and page access based on the authenticated user's role, ensuring that users only see functionalities relevant to their role. Unauthorized users attempting to access restricted resources are redirected to an access-denied

page with proper logging of the event.

Additionally, **regular database backups** are scheduled to prevent data loss. Backup files are stored securely and can be restored in case of server failure or disaster recovery scenarios.

Overall, the Partner Management System adopts a **multi-layered security architecture**, combining secure coding practices, role-based access control, encrypted communication, real-time activity logging, and proactive threat mitigation strategies to ensure the confidentiality, integrity, and availability of the application and its data.

CHAPTER NO. 9

9.1 LIMITATION

Despite the robust security measures and system functionalities implemented in the Partner Management System, there are a few limitations that should be considered. Firstly, the system's performance may degrade under heavy traffic or simultaneous large-scale file uploads, as it currently lacks advanced scaling features to handle high user volumes effectively. Additionally, while data encryption is implemented, there could be potential vulnerabilities if new, unknown attack vectors arise, as security needs to evolve constantly to combat emerging threats. The system also relies heavily on third-party services for notifications and document validation, which could pose integration challenges or disruptions in case of service downtimes.

Another limitation is that the security measures, while comprehensive, may not account for all possible threat scenarios, such as zero-day vulnerabilities or sophisticated social engineering attacks. User experience can also be impacted by security measures, like multi-factor authentication (MFA), which, while adding a layer of security, could create barriers for less tech-savvy users. Lastly, the system currently has limited provisions for handling large-scale data analytics or complex report generation, which could be crucial as the system scales.

While the Partner Management System fulfills its core objectives of streamlining partner registration, offer management, and role-based interactions (admin, partner, user, shopkeeper, and employee), there are certain limitations that affect its scalability and flexibility in a real-world enterprise environment.

Key Limitations Identified:

1. **Limited File Validation**
   Although document uploads for Aadhar and PAN are supported, advanced file validation like **OCR-based verification** or **digital signature checks** is not yet implemented. This can affect document authenticity in high-security environments.
2. **Email Notification Reliability**
   The system supports basic email or in-app notifications for offer approvals and rejections. However, it may lack **high reliability under large-scale traffic** or when **SMTP servers** face delays or downtime, which can affect timely communication.
3. **No Mobile App Support**
   The application is currently **web-based only**. Lack of a dedicated **mobile application** limits its accessibility for shopkeepers or employees who may prefer using smartphones for quick QR code scanning or viewing offers.
4. **QR Code Security**
   Although **unique QR codes** are generated for each approved offer, they can potentially be reused or screenshotted without additional verification layers like **geolocation validation**, **device binding**, or **expiration timers**.
5. **Limited Analytics and Reporting**
   The admin panel currently focuses on partner and offer management, but lacks **advanced analytics features** such as **usage trends**, **real-time dashboards**, or **partner engagement insights**.

Role Expansion Challenges

Adding new roles or dynamically adjusting permissions requires code-level changes, as the current **role-based access control** is **statically defined** during development and not designed for runtime configuration.

6. **Error Recovery**
   While **basic error handling and logging** are in place using Serilog, the system does not support **automated recovery mechanisms** like **retry queues**, **transaction rollbacks**, or **failure fallback logic**, which are essential in enterprise-grade systems.
7. **Scalability Concerns**
   The system is designed for **small to medium-scale deployments**. For enterprise-scale scenarios involving **thousands of concurrent users**, enhancements like **load balancing**, **distributed architecture**, or **cloud-native deployment** would be necessary to maintain performance and reliability.

Despite these limitations, the system serves as a solid foundation for managing partner engagement and offer workflows. Future enhancements could include mobile-first design, OTP-secured QR scanning, AI-based offer recommendations, dynamic role configuration, and improved analytics dashboards, making the Partner Management System even more powerful and enterprise-ready.

CHAPTER NO. 10

10.1CONCLUSION AND FUTURE SCOPE

The Partner Management System has been successfully designed and implemented to streamline the partner registration process, offer management, and ensure secure role-based access for users, partners, employees, and administrators. The system integrates essential features like document upload, offer approval/rejection workflows, QR code validation, and automated notifications, providing a seamless and user-friendly experience. Robust security measures such as encryption, input validation, role-based access, and session management have been implemented to safeguard sensitive data and prevent unauthorized access. With thorough testing, debugging, and validation, the system is expected to meet user requirements and function reliably in a real-world environment.

The development of the Partner Management System marks a significant step toward digitizing and streamlining the complex workflow of managing partners, offers, and their interaction with employees, users, and shopkeepers. This project successfully integrates multiple modules such as role-based authentication, partner onboarding with document uploads, admin-controlled offer approval and rejection, and QR code-based offer validation, thereby enhancing transparency, operational control, and user engagement.

The system provides a centralized platform where partners can register and submit offers, which are then carefully reviewed by admins for legitimacy and alignment with organizational goals. Once approved, these offers become accessible to employees, who can view, use, and track them through dynamic QR codes. Shopkeepers validate these offers, ensuring that once redeemed, the offer cannot be reused—thus preserving integrity. The addition of real-time notifications, structured user dashboards, and clean UI design further improves the user experience for all stakeholders involved.

From a technical standpoint, the project showcases efficient use of ASP.NET Core MVC, SQL Server, and tools like Serilog for logging, along with a responsive front-end built with modern design practices. The system promotes security by restricting access based on roles, validating input data, and logging critical operations.

Although the project has some limitations—such as the lack of a mobile version, limited analytics, and scalability constraints—it establishes a strong foundation for future improvements. With further development, features like AI-driven offer suggestions, OCR-based document verification, and a mobile app could greatly enhance its utility and reach.

In conclusion, the Partner Management System achieves its primary objective of automating and simplifying partner-offer-user interactions. It stands as a robust, user-

friendly, and scalable solution that not only reduces manual effort but also ensures better management, tracking, and accountability in partner-related operations.

10.2 Future Scope

While the current implementation fulfils the essential requirements of the Partner Management System, there is significant scope for future enhancements. One key area for improvement is scalability—optimizing the system to handle higher traffic and larger datasets with improved performance. Future versions could integrate advanced analytics for better tracking of offers, user behaviour, and system performance. Adding more robust AI-powered features, such as predictive analytics for offer trends or personalized recommendations for partners, could enhance user experience and business insights.

Additionally, expanding the system to support mobile platforms (iOS/Android) could improve accessibility and user engagement. The incorporation of a more sophisticated reporting and dashboard system would allow better insights for administrators and partners. Lastly, adopting advanced security features like biometric authentication or machine learning-based fraud detection could further strengthen the security of the system and reduce the risk of malicious activities. With these advancements, the Partner Management System can continue to evolve and meet the growing needs of its users.

Key Areas for Future Development:

1) **Mobile Application Integration**

   To improve accessibility and user convenience, especially for shopkeepers and employees, a dedicated mobile application (Android and iOS) can be developed. This app would allow users to register, access offers, scan QR codes, and track their transactions seamlessly, even without a desktop system.

2) **AI & Machine Learning Integration**

   Future versions can integrate AI-based recommendation systems to suggest personalized offers to employees based on usage history, preferences, or location. Admins could also use predictive analytics to identify high-performing partners or trending offers.

3) **Advanced Document Verification**

   Enhance document upload features by integrating OCR (Optical Character Recognition) and e-KYC systems to verify Aadhar and PAN details automatically. This would reduce manual verification effort and increase security and accuracy.

4) **Real-time Offer Tracking & Analytics**

The admin panel can be upgraded with detailed dashboards that show real-time analytics on partner activity, offer performance, redemption rates, regional trends, and engagement metrics. These insights will help make informed decisions.

5) **Enhanced QR Code Security**

To improve security, time-bound QR codes, device-specific access, and OTP-based validation can be implemented. These features will help ensure that offers cannot be misused or fraudulently duplicated.

6) **Multilingual Support**

Adding multilingual capability will make the system more accessible to a wider audience, including non-English speaking partners and users across India or other regions.

7) **Cloud Deployment & Scalability**

The current system can be migrated to cloud platforms like Microsoft Azure or AWS, allowing for auto-scaling, high availability, and data redundancy, making it suitable for large-scale enterprise use.

8) **Role Customization and Permissions**

A more dynamic and customizable role management module can be added where admins can create and assign custom roles with specific permissions, improving flexibility across organizations.

9) **Automated Workflow and Notifications**

Implementing automated workflows where offers move through various approval levels or expiry checks without manual intervention, along with scheduled alerts and SMS notifications, can enhance operational efficiency.

10) **Third-party Integrations**

Integration with third-party services such as payment gateways, CRM tools, and ERP systems will increase the system's utility in large business ecosystems.

REFERENCES

Oracle. (2023). *The Java™ Tutorials*. Retrieved from **https://docs.oracle.com/javase/tutorial/**
Pivotal Software, Inc. (2023). *Spring Boot Reference Documentation*. Retrieved from

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
MongoDB Inc. (2023). *MongoDB Atlas Documentation*. Retrieved from
https://www.mongodb.com/docs/atlas/

Google. (2023). *Flutter Documentation*. Retrieved from https://flutter.dev/docs
Dart Team. (2023). *Dart Programming Language Documentation*. Retrieved from
https://dart.dev/guides

Baeldung. (2023). *Spring Boot Tutorials*. Retrieved from https://www.baeldung.com/spring-boot-start

Firebase. (2023). *Using Firebase for Real-Time Chat in Flutter*. Retrieved from
https://firebase.google.com/docs/flutter/setup

Stack Overflow Community. (2023). *Programming Q&A Help*. Retrieved from
https://stackoverflow.com/

Medium Articles. (2023). *Building Chat App with Flutter and WebSocket*. Retrieved from
https://medium.com/

GitHub Repositories. (2023). *Open-source Flutter Chat Projects for Learning*. Retrieved from