

A MINI PROJECT REPORT ON

CHATting APPLICATION IN JAVA

USING SOCKET PRIGRAMMING

SUBMITTED TOWARDS THE
PARTIAL FULFILLMENT FOR
V SEMESTER OF

Diploma In Engineering (Information Technology)

BY

Name	Roll no.
Aryan Shivaji Bhopale	216004
Rudraksh Nilesh Ghodake	216017

Under The Guidance of

M.B. patil sir



DEPARTMENT OF INFORMATION TECHNOLOGY
Government Polytechnic Kolhapur
(An Autonomous Institute of Government of Maharashtra)
University Road, Vidyanagar, Kolhapur – 416004 (Maharashtra)



**GOVERNMENT POLYTECHNIC KOLHAPUR
DEPARTMENT OF INFORMATION TECHNOLOGY**

CERTIFICATE

This is to certify that the Micro Project Report Entitled

**CHATting APPLICATION IN JAVA
USING SOCKET PRIGRAMMING**

Submitted by

Name	Roll no.
Aryan Shivaji Bhopale	216004
Rudraksh Nilesh Ghodake	216017

successfully completed above entitled Micro-Project in the Course of **ITG305-Database Management System** in the **III** semester during his/her tenure of completing the Diploma in Information Technology

Prof.Mr.S J Pukale
Course Inchrge
Dept. of Information Tech.

Prof.S.A.Nadgeri
H.O.D
Dept. of Information Tech.

Abstract

This project focuses on creating a real-time chat application using Java and socket programming. This project aims to provide a platform for users to exchange messages, fostering communication and collaboration over the internet.

Introduction: With the advent of digital communication, chat applications have become an integral part of our daily lives. This project introduces a Java-based chat application that leverages socket programming to facilitate real-time text-based conversations. Socket programming allows for direct communication between two or more devices over a network, making it a crucial component of the internet's backbone.

Motivation: The motivation behind this project is to understand and implement the fundamental concepts of network communication and socket programming while creating a practical, user-friendly chat application. Additionally, it serves as an opportunity to explore various aspects of software development, such as user interfaces, client-server architecture, and message handling, which are crucial skills for software engineers.

Outcome: The key outcomes of this project include:

1. A functional chat application that allows users to create accounts, log in, and send messages to other users in real time.
2. A deeper understanding of socket programming principles, network communication, and data serialization.
3. Enhanced skills in Java programming, user interface design, and database management.
4. Improved knowledge of client-server architecture and the implementation of a multi-threaded server to handle multiple client connections simultaneously.
5. A valuable resource for developers and students interested in learning about network programming and chat application development.

Innovation: While this project primarily focuses on implementing fundamental socket programming techniques for chat application development, it may incorporate innovative features, such as encryption for enhanced security, multimedia file sharing, or integration with other platforms or services. The innovative aspect may depend on the specific goals and requirements of the project.

In conclusion, the development of a chat application in Java using socket programming is a valuable project that combines essential programming concepts with practical applications. It provides a platform for real-time communication, enhances software development skills, and opens doors to potential innovations in the realm of chat applications.

INDEX

1	Micro-Project Proposal	1
1.1	Aims/Benefits of the Micro-Project	2
1.2	Course Outcomes Addressed	2
2	Proposed Methodology	3
2.1	Action Plan	4
2.2	Resources Required	4
3	Architecture	5
3.1	Setup/Diagram /Flowchart	6
4	Program/Circuit Diagram	7
5	Output/Results	8
6	Applications of this Micro-Project	9
7	References	11
	Annexure A Assessment Scheme	13

CHAPTER 1

MICRO-PROJECT PROPOSAL

1.1 AIMS/BENEFITS OF THE MICRO-PROJECT

A micro-project on creating a chat application in Java using socket programming offers several aims and benefits, both in terms of technical and practical learning. Here are some of the key aims and benefits of such a project:

1. **Hands-on Experience:** Developing a chat application through socket programming provides hands-on experience in building networked applications. This practical knowledge can be valuable for students and developers interested in understanding how network communication works.
2. **Understanding Socket Programming:** It helps in gaining a deep understanding of socket programming concepts, such as TCP and UDP protocols, socket creation, binding, and data transfer. These skills are transferable to various network-related tasks.
3. **Client-Server Architecture:** Learning to create a chat application necessitates understanding client-server architecture, which is a fundamental concept in networking. This knowledge is applicable to many other networked applications beyond chat.
4. **Multi-Threading:** Implementing a multi-threaded server to handle multiple clients concurrently is a crucial skill. It teaches how to manage concurrency, preventing bottlenecks in server performance.
5. **User Interface Design:** Designing a user-friendly interface for the chat application enhances skills in GUI (Graphical User Interface) design. This is valuable for developers interested in creating visually appealing and user-centric applications.
6. **Database Integration:** Depending on the project's complexity, incorporating user account management or message storage may require interaction with databases. This can provide experience in database integration and management.
7. **Real-Time Communication:** Building a real-time chat application allows developers to explore real-time communication challenges, such as message synchronization and updates, which are crucial in various applications, including messaging apps and collaboration tools.
8. **Security Considerations:** Understanding and implementing security measures, such as data encryption, user authentication, and secure data transfer, can be a part of the project, helping learners grasp essential security concepts.
9. **Practical Application:** A chat application serves as a practical project with real-world relevance. It can be used for personal communication or even extended for use within a small group, making it immediately useful.
10. **Innovation and Customization:** Developers have the opportunity to innovate by adding unique features to the chat application. This customization can include additional functionalities, integration with other services, or enhanced user experience features.
11. **Portfolio Enhancement:** Completing a micro-project like this can be a valuable addition to one's portfolio, showcasing the ability to develop networked applications and software development skills.

12. **Collaborative Skills:** If the project is executed in a team, it offers an opportunity to improve collaborative skills, such as project management, code version control, and teamwork.

In summary, a micro-project on a chat application in Java using socket programming is a valuable educational endeavor that combines various technical and practical benefits, from deepening technical skills to enhancing real-world application development capabilities.

1.2 Course Outcomes Addressed

A project on developing a chat application in Java using socket programming can address several course outcomes, depending on the specific goals and objectives of the course. Below are some common course outcomes that such a project can help address:

1. ****Understanding of Socket Programming**:**
 - Outcome: Students will demonstrate a thorough understanding of socket programming concepts, including socket creation, binding, and communication.
2. ****Network Communication**:**
 - Outcome: Students will be able to explain and apply network communication protocols, such as TCP/IP or UDP, and understand the differences between them.
3. ****Client-Server Architecture**:**
 - Outcome: Students will comprehend the principles of client-server architecture and apply them to design and implement a functional chat application.
4. ****Multi-Threading**:**
 - Outcome: Students will demonstrate the ability to design and implement multi-threaded server and client applications to handle multiple concurrent connections.
5. ****User Interface Design**:**
 - Outcome: Students will be capable of designing user-friendly graphical user interfaces (GUI) for their chat application using Java's GUI libraries (e.g., Swing or JavaFX).
6. ****Database Integration** (if applicable):**
 - Outcome: Students will understand how to integrate a database system for user account management, message storage, or other data-related tasks within the chat application.
7. ****Real-Time Communication**:**
 - Outcome: Students will have a grasp of real-time communication challenges and be able to implement features for message synchronization, updates, and notifications.
8. ****Security Principles** (if applicable):**
 - Outcome: Students will comprehend security principles related to networked applications, including user authentication, data encryption, and secure data transfer.
9. ****Error Handling and Exception Handling**:**
 - Outcome: Students will be proficient in handling errors and exceptions that may arise during

network communication, such as connection failures or data transmission issues.

10. ****Project Management and Documentation****:

- Outcome: Students will develop skills in project management, code documentation, and the ability to provide clear and concise project documentation, including user manuals or technical reports.

11. ****Testing and Debugging****:

- Outcome: Students will learn to test and debug their chat application effectively, identifying and addressing issues that may arise during development.

12. ****Innovation and Customization****:

- Outcome: Students will showcase innovation by adding unique features or customizations to the chat application, demonstrating creativity and problem-solving skills.

13. ****Collaborative Skills**** (if applicable):

- Outcome: If the project is executed in teams, students will enhance their collaborative skills, including teamwork, communication, and coordination.

14. ****Problem Solving and Troubleshooting****:

- Outcome: Students will develop problem-solving skills and the ability to troubleshoot and resolve issues related to the chat application and network communication.

15. ****Presentation Skills**** (if required):

- Outcome: Students may be required to present their project to peers or instructors, improving their presentation and communication skills.

By addressing these course outcomes, students will gain a comprehensive understanding of network programming, software development, and related skills, making them well-prepared for future projects and careers in software development and networking.

CHAPTER 2

PROPOSED METHODOLOGY

2.1 ACTION PLAN

Creating an action plan for a project like a chat application in Java using socket programming is essential for organizing your work and ensuring a smooth development process. Below is a detailed action plan to guide you through the project:

****1. Define Project Scope and Objectives:****

- Clearly outline the goals and objectives of the project.
- Determine the specific features and functionalities you want to implement in the chat application.

****2. Requirements Analysis:****

- Identify user requirements, such as user account creation, real-time messaging, and any additional features.
- Document technical requirements, including the choice of programming languages, libraries, and tools.

****3. Design Phase:****

- Create a high-level system architecture and design the chat application's components.
- Develop wireframes or mock-ups for the user interface.
- Decide on the user interface design framework (Swing, JavaFX, etc.).
- Plan the database schema (if needed) for user accounts and message storage.

****4. Set Up the Development Environment:****

- Install the required development tools and libraries (e.g., Java Development Kit, IDEs, GUI libraries).
- Set up version control (e.g., Git) and create a repository to manage the project's source code.

****5. Socket Programming Basics:****

- Study socket programming concepts and networking fundamentals.
- Learn about the Java Socket API for creating and managing sockets.

****6. Create the Server Component:****

- Develop the chat server using Java, considering multi-threading for handling multiple clients.
- Implement socket creation, binding, and listening for incoming connections.
- Develop the server logic for message routing and handling.

****7. Create the Client Component:****

- Build the chat client application, including the user interface.
- Implement socket connections to the server and message sending/receiving.
- Integrate user authentication (if required).

****8. Real-Time Communication:****

- Implement real-time message updates and notifications using sockets.
- Develop mechanisms for handling incoming messages and displaying them in the user interface.

****9. User Account Management (if applicable):****

- Implement user registration and login features.
- Set up a database (e.g., SQLite, MySQL) to store user account information.

****10. Security and Encryption (if applicable):****

- Add security measures, such as data encryption and user authentication, to protect the chat

application.

****11. Testing and Debugging:****

- Conduct extensive testing to identify and fix any bugs or issues in the application.
- Test the application's functionality under various scenarios.

****12. Documentation:****

- Create user manuals, installation guides, and technical documentation for the project.
- Document the source code with comments for clarity and maintainability.

****13. Innovation and Customization (if desired):****

- Implement any unique features or customizations that set your chat application apart.

****14. Project Presentation (if required):****

- Prepare a presentation or demonstration of the chat application to showcase its features and functionality.

****15. Project Review and Refinement:****

- Review the project for any areas that need improvement or optimization.
- Incorporate feedback from peers, instructors, or users (if available).

****16. Finalization and Deployment:****

- Make any necessary refinements to the application.
- Prepare the application for deployment.

****17. Project Evaluation:****

- Evaluate the project's success against the initial objectives and requirements.

****18. Post-Project Maintenance (optional):****

- Consider ongoing maintenance and support, addressing any issues that may arise.

****19. Knowledge Transfer:****

- Share your knowledge and experience with others who may benefit from the project's insights and code.

Remember that project management is iterative, and you may need to revisit and adjust various phases as the project progresses. Proper planning and documentation are key to the success of your chat application project in Java using socket programming.

2.2 RESOURCES REQUIRED

When undertaking a project to develop a chat application in Java using socket programming, you will require various resources to ensure its successful completion. Here is a list of essential resources:

1. **Hardware Resources:**

- **Computer:** You'll need a computer to develop, test, and run the chat application.
- **Internet Connection:** An internet connection is required for testing the application's network functionality.

2. **Software Resources:**

- ****Java Development Kit (JDK):**** To write and compile Java code.
- ****Integrated Development Environment (IDE):**** Choose a Java IDE like Eclipse, IntelliJ

IDEA, or NetBeans for coding and debugging.

- **Version Control System:** Set up Git or another version control system for tracking code changes.
- **Database Management System (if needed):** Software like MySQL, PostgreSQL, SQLite, or H2 for database operations.
- **GUI Toolkit:** If you're creating a graphical user interface, you'll need a Java GUI library like Swing or JavaFX.
- **Socket Programming Libraries:** Java provides the `java.net` package for socket programming.
- **Operating System:** Ensure your development environment is compatible with your chosen OS (Windows, macOS, Linux).
- **Documentation Tools:** Software for creating user manuals, technical documentation, and project reports (e.g., Microsoft Word, LaTeX).
- **Collaboration Tools:** Communication and collaboration tools for working in a team (e.g., Slack, Microsoft Teams).

3. **Programming Skills and Knowledge:**

- A strong understanding of Java programming is necessary.
- Familiarity with socket programming concepts and network protocols (TCP, UDP).
- Knowledge of database management (if you plan to include user account management or message storage).

4. **Development Environment Setup:**

- Configure your development environment with the necessary plugins and extensions for your IDE.
- Ensure that your IDE is compatible with the chosen Java version.

5. **Server Hosting (if deploying the application):**

- If you plan to deploy the chat server to a remote host, you will need access to a server or hosting service.

6. **Learning Resources:**

- Online tutorials, books, and courses on Java programming, socket programming, and GUI development can be valuable resources for gaining knowledge and solving challenges.

7. **Team Collaboration (if working in a team):**

- Tools for team communication and collaboration, like version control (e.g., Git), project management (e.g., Jira, Trello), and instant messaging or video conferencing (e.g., Slack, Microsoft Teams).

8. **Documentation and Reporting Tools:**

- Tools for creating project documentation, user manuals, and reports (e.g., Microsoft Word, Google Docs, LaTeX).

9. **Security Tools (if implementing security features):**

- For implementing and testing security measures like encryption and authentication, you may need security-related tools and libraries.

10. **Testing and Debugging Tools:**

- Debugging tools and testing frameworks for identifying and resolving issues in the application.

11. **User Testing and Feedback:**

- If possible, access to potential users for user testing and feedback.

12. ****Innovation and Customization Resources (if applicable):****

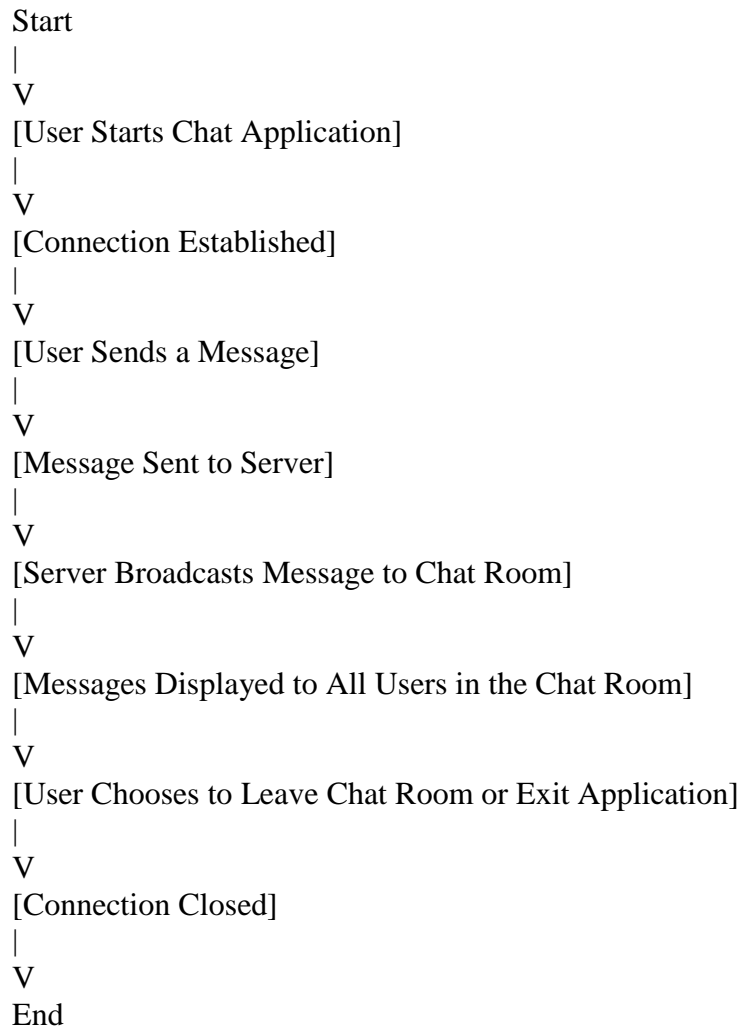
- Depending on your project's goals, you may need resources related to innovative features, such as multimedia processing or integration with external services or APIs.

Remember that the specific resources required may vary depending on the project's complexity and objectives. It's important to plan and gather the necessary resources to ensure a smooth and successful development process for your chat application in Java using socket programming.

CHAPTER 3

ARCHITECTURE

3.1 FLOWCHART



CHAPTER 4:

PROGRAM

Server side:

```
package chatting.application;

import static chatting.application.Client.a1;

import javax.swing.*.*;

import javax.swing.border.*;

import java.awt.*.*;

import java.awt.event.*;

import java.util.*;

import java.text.*;

import java.net.*;

import java.io.*;

public class Server implements ActionListener{

    JTextField text;

    JPanel a1;

    static Box vertical = Box.createVerticalBox();

    static JFrame f = new JFrame();

    static DataOutputStream dout;

    Server(){

        f.setLayout(null);

        JPanel p1 = new JPanel();

        p1.setBackground(new Color(7,94,84 ));

        p1.setBounds(0,0,350,55);

        f.add(p1);

        p1.setLayout(null);

        ImageIcon i1 = new ImageIcon(ClassLoader.getResource("icons/3.png"));

        Image i2 = i1.getImage().getScaledInstance(20,20,Image.SCALE_DEFAULT);

        ImageIcon i3 = new ImageIcon(i2);

        JLabel back = new JLabel(i3);

        back.setBounds(5,20,20,20);

        p1.add(back);

        back.addMouseListener(new MouseAdapter(){

            public void mouseClicked(MouseEvent ae){

                System.exit(0);

            }

        });

    }

}
```

```
ImageIcon i4 = new ImageIcon(ClassLoader.getResource("icons/1.rudra.jpg"));
Image i5 = i4.getImage().getScaledInstance(40,40,Image.SCALE_DEFAULT);
ImageIcon i6 = new ImageIcon(i5);
JLabel profile = new JLabel(i6);
profile.setBounds(40,10,40,40);
p1.add(profile);
```

```
ImageIcon i7 = new ImageIcon(ClassLoader.getResource("icons/video.png "));
Image i8 = i7.getImage().getScaledInstance(25,25,Image.SCALE_DEFAULT);
ImageIcon i9 = new ImageIcon(i8);
JLabel video = new JLabel(i9);
video.setBounds(230,15,25,25);
p1.add(video);
```

```
ImageIcon i10 = new ImageIcon(ClassLoader.getResource("icons/phone.png "));
Image i11 = i10.getImage().getScaledInstance(25,25,Image.SCALE_DEFAULT);
ImageIcon i12 = new ImageIcon(i11);
JLabel phone = new JLabel(i12);
phone.setBounds(280,15,25,25);
p1.add(phone);
```

```
ImageIcon i13 = new ImageIcon(ClassLoader.getResource("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(320,15,10, 25);
p1.add(morevert);
```

```
JLabel name = new JLabel("Rudraksh");
name.setBounds(90,15,100,15);
name.setForeground(Color.WHITE);
name.setFont(new Font("SAN_SERIF",Font.BOLD,15));
p1.add(name);
```

```
JLabel status = new JLabel("Active Now");
status.setBounds(90,30,100,15);
status.setForeground(Color.WHITE);
status.setFont(new Font("SAN_SERIF",Font.BOLD,10));
```

```

p1.add(status);

a1 = new JPanel();
a1.setBounds(0,50,350 ,465);
f.add(a1);


text = new JTextField();
text.setBounds(0,515,250,30 );
text.setFont(new Font("SAN_SERIF",Font.PLAIN,12));
f.add(text);


JButton send = new JButton("Send");
send.setBounds(250,515,100,30);
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(350,550);
f.setLocation(200,50);
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(13));
```

```
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:125px\">" + out + "</p></html>");
```

```
    output.setFont(new Font("Tahoma",Font.PLAIN,14));
```

```
    output.setBackground(new Color(37,211,102));
```

```
    output.setOpaque(true);
```

```
    output.setBorder(new EmptyBorder(13,13,13,45));
```

```
    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();

        }

    }

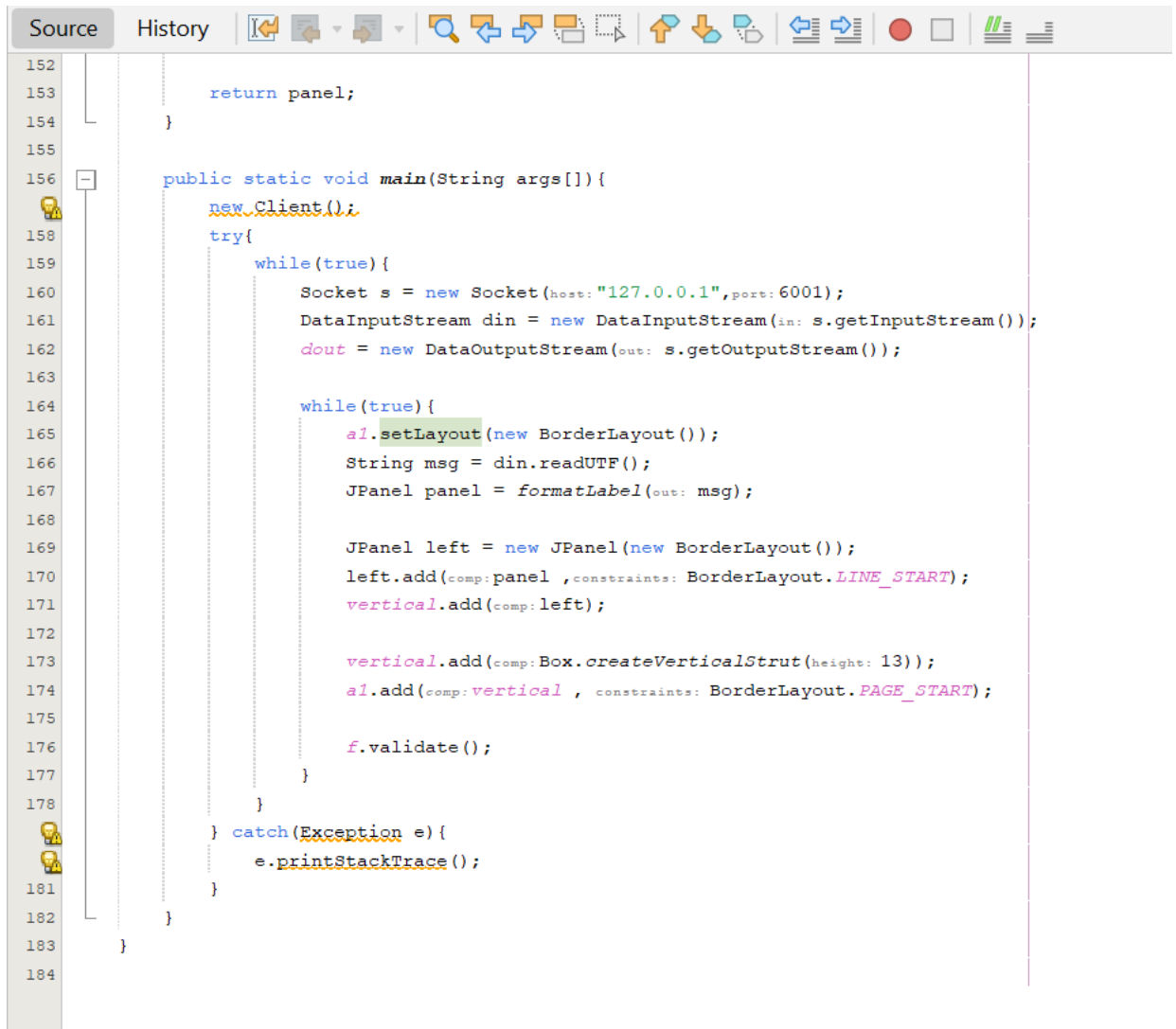
} catch(Exception e){

    e.printStackTrace();

}

}
```

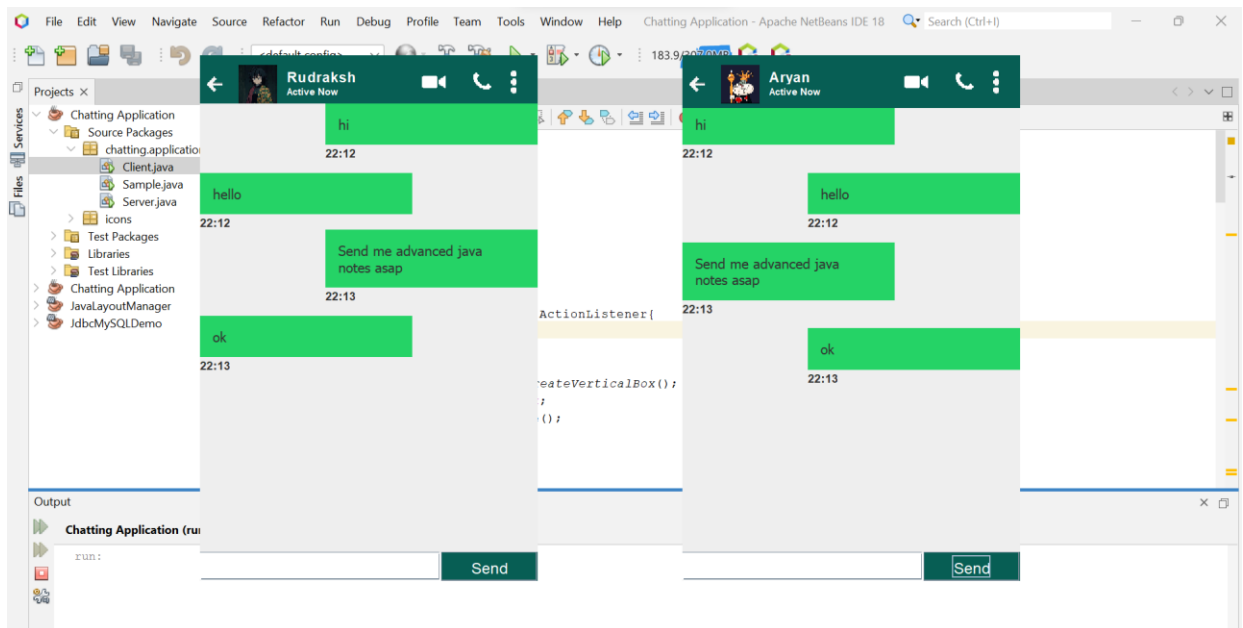
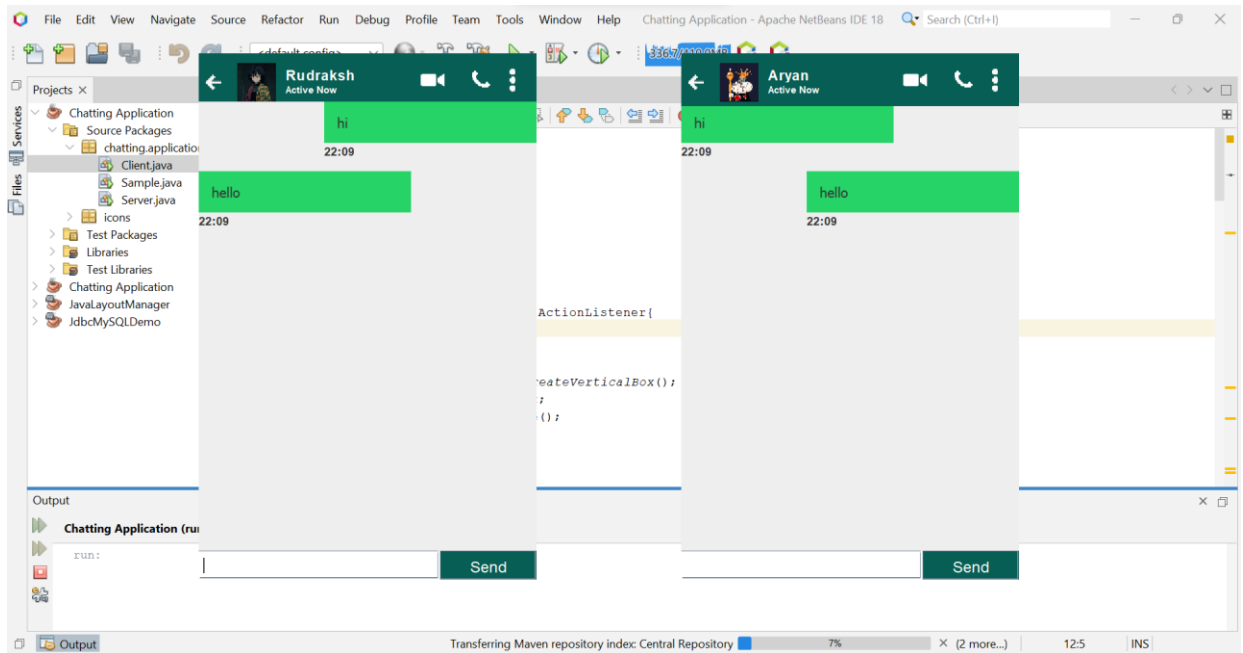
Client side :



```
152
153     return panel;
154 }
155
156 public static void main(String args[]){
157     new Client();
158     try{
159         while(true){
160             Socket s = new Socket(host: "127.0.0.1",port: 6001);
161             DataInputStream din = new DataInputStream(in: s.getInputStream());
162             dout = new DataOutputStream(out: s.getOutputStream());
163
164             while(true){
165                 a1.setLayout(new BorderLayout());
166                 String msg = din.readUTF();
167                 JPanel panel = formatLabel(out: msg);
168
169                 JPanel left = new JPanel(new BorderLayout());
170                 left.add(comp: panel ,constraints: BorderLayout.LINE_START);
171                 vertical.add(comp: left);
172
173                 vertical.add(comp: Box.createVerticalStrut(height: 13));
174                 a1.add(comp: vertical , constraints: BorderLayout.PAGE_START);
175
176                 f.validate();
177             }
178         }
179     } catch(Exception e){
180         e.printStackTrace();
181     }
182 }
183
184 }
```


CHAPTER 5

OUTPUT/RESULTS



CHAPTER 6

APPLICATIONS OF THIS

MICRO-PROJECT

A micro-project on a chatting application in Java using socket programming can have various practical applications and can be used for educational, professional, and personal purposes. Here are some of the applications of such a project:

1.	Learning and Skill Development:
	<ul style="list-style-type: none">• Education: Students can use the project as a practical way to learn about socket programming, Java, client-server architecture, and network communication.• Skill Enhancement: Developers can use it to improve their programming, network, and software development skills.
2.	Collaboration and Communication:
	<ul style="list-style-type: none">• Team Communication: Small teams or organizations can deploy a chat application for internal communication, making it easier for team members to collaborate in real-time.• Customer Support: Businesses can use a chat application for providing customer support, allowing customers to chat with support agents.
3.	Academic Projects:
	<ul style="list-style-type: none">• Academic Assignments: Students can use the chat application as part of their academic projects, demonstrating their understanding of networking and programming concepts.
4.	Prototyping and Proof of Concept:
	<ul style="list-style-type: none">• Proof of Concept: Developers can use the project as a starting point to demonstrate the feasibility of implementing real-time communication features in their applications.• Prototyping: Startups and entrepreneurs can use a basic chat application as a prototype for more complex communication or social networking apps.
5.	Hackathons and Coding Competitions:
	<ul style="list-style-type: none">• As a Challenge: Chat applications can be used as challenges or tasks in hackathons or coding competitions to test participants' problem-solving skills.
6.	Personal Use and Hobbies:
	<ul style="list-style-type: none">• Personal Chat: Individuals can use a self-built chat application for private conversations with friends and family.• Hobbies: Enthusiasts can create custom chat applications with unique features or themes as a hobby project.
7.	Innovation and Integration:
	<ul style="list-style-type: none">• Innovation: Developers can add innovative features to the chat application, such as multimedia sharing, bots, or integration with other services.• Integration: Businesses can integrate a chat application into their websites or mobile apps to provide live chat support or enhance user engagement.
8.	Educational Demonstrations:
	<ul style="list-style-type: none">• Teaching Tool: Educators can use the chat application as a teaching tool to explain networking and software development concepts to their students.
9.	Open Source Projects:

	<ul style="list-style-type: none"> • Contribution: Developers can contribute their chat application project to open-source communities, allowing others to benefit and collaborate on its development.
10.	Testing and Quality Assurance:
	<ul style="list-style-type: none"> • Testing Ground: Quality assurance teams can use the chat application to test network-related aspects of software or web applications.
11.	Community Building:
	<ul style="list-style-type: none"> • Online Communities: Online forums, social groups, or communities can use chat applications to foster discussions and community engagement.
12.	Fun and Entertainment:
	<ul style="list-style-type: none"> • Gaming: Gamers can use chat applications within online games for real-time communication with other players. • Fun Projects: Individuals or groups can create unique, themed chat applications for fun or entertainment.

The applications of a chat application project are diverse, ranging from educational and professional uses to personal hobbies and innovative ideas. Depending on the specific features and customization, the project can be tailored to suit various needs and scenarios.

ANNEXURE A

ASSESSMENT SCHEME

Sr No	Assessment Parameter	Out of Marks	Obtained Marks
1	Technical preparedness for practical	5	
2	Operating skills/Algorithm/ flowchart	5	
3	Observation/Logic/ Program	5	
4	Results/Output	5	
5	Safety/Discipline and Punctuality	5	
6	Total	Out of 25	Signature with Date