**KARNATAKA LAW SOCIETY’S**

**GOGTE INSTITUTE OF TECHNOLOGY**

UDYAMBAG, BELAGAVI-590008

(An Autonomous Institute under Visveswaraya Technological University, Belgaum)

**(APPROVED BY AICTE, NEW DELHI)**



Course Seminar Report on

**“BANKING SYSTEM”**

Submitted in the partial fulfilment for the academic requirements of

Fifth Semester B.E

Information Science and Engineering

Submitted by:

|  |  |
| --- | --- |
| NAME | USN |
| Ayush anand | 2GI21IS013 |
| Raghupraveer Nippani | 2GI21IS065 |
| Nandeesh Hiremath | 2GI21IS035 |

Guidance of

Dr. Padma Dandannavar.

PROFESSOR, Dept of ISE

2023-2024

**CERTIFICATE**

This is to certify that the Seminar entitled “**Banking System**” is a bonafide record of the Seminar work done by **Ayush anand(2GI21IS013), Raghupraveer Nippani(2GI21IS065) and Nandeesh Hiremath (2GI21IS035)** under the supervision and guidance of **Dr. Padma Dandannavar**, in partial fulfilment of the requirements for the Outcome Based Education Paradigm in ISE from KLS Gogte Institute of Technology for the academic year 2023-2024

Faculty in charge Head of Department

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Topic** | **Page** |
| 1 | Abstract | 4 |
| 2 | Introduction | 5 |
| 3 | Objectives | 6 |
| 4 | Methodology | 7 |
| 5 | Code Explanation | 8 |
| 6 | Results | 10 |
| 7 | Conclusion | 12 |
| 8 | Reference | 13 |

**Abstract:**

This project presents the design and implementation of a distributed banking system utilizing Remote Method Invocation (RMI) technology. The objective is to create a scalable and secure architecture where clients can perform banking operations such as deposit, withdrawal, and balance inquiry through interactions with a central server. The application adheres to a client-server model, allowing users to access banking services over a network.

The project employs Java RMI for remote communication, enabling seamless invocation of methods on the server from client applications. The client-server architecture ensures separation of concerns, with the server managing account-related operations and the client interacting with the user interface. Key features include account creation, deposit, withdrawal, and balance inquiry.

The report outlines the system's architecture, detailing the design choices made to achieve distributed functionality. It covers the implementation of the server-side logic for account management and the client-side application for user interaction. Emphasis is placed on the use of synchronization for concurrent access to shared resources and error handling for robustness.

Throughout the project, considerations for security, reliability, and scalability are addressed. The application's extensibility is demonstrated through the ability to handle multiple client connections concurrently. Challenges faced and solutions devised during the development process are discussed.

In conclusion, this distributed banking system provides a foundation for secure and scalable financial transactions over a network, showcasing the effectiveness of Java RMI in facilitating remote communication between client and server components. The report encapsulates the project's goals, methodologies, and outcomes, offering insights into the design and implementation of the distributed banking application.

**Introduction:**

In today's interconnected world, the design and implementation of distributed systems play a pivotal role in addressing the demands of modern applications. This project introduces a distributed banking system, leveraging the power of Remote Method Invocation (RMI) to create a robust, scalable, and secure architecture for managing financial transactions.

**Background:**

The financial sector constantly evolves to meet the growing demands of users seeking efficient and secure banking services. Distributed systems offer a solution to address the challenges associated with providing real-time access to banking functionalities from diverse locations.

**Objectives:**

The primary objectives of Banking System are as follows:

**1**. **Client-Server Architecture Implementation:**

- Develop a robust client-server architecture to facilitate communication between multiple clients and a centralized server.

**2.RMI Integration:**

- Incorporate Java RMI technology to enable seamless remote method invocation, allowing clients to perform banking operations on the server.

**3. Account Management Functionality:**

- Implement essential banking functionalities on the server, including account creation, deposit, withdrawal, and balance inquiries.

**4. User Interface Design:**

- Design an intuitive user interface for clients to interact with the distributed banking system, ensuring a user-friendly experience.

**5.Concurrency and Thread Safety:**

- Implement synchronization mechanisms to address concurrent access issues, ensuring thread safety in shared resource manipulation.

**6. Error Handling and Robustness:**

- Develop a robust error-handling mechanism to manage exceptional scenarios gracefully and enhance the overall reliability of the distributed application.

**7. Scalability Testing:**

- Conduct scalability testing to assess the system's performance under varying loads, ensuring it can handle multiple client connections concurrently.

**8. Security Measures:**

- Incorporate security measures to safeguard financial transactions, considering encryption and authentication protocols to protect sensitive data.

**9. Extensibility and Adaptability:**

- Design the distributed banking system with extensibility in mind, allowing for the addition of new features and adaptations to future changes in financial requirements.

**10. Documentation and Reporting:**

- Provide comprehensive documentation covering the system's architecture, implementation details, usage instructions, and insights gained during the development process.

**11. User Acceptance Testing:**

- Conduct user acceptance testing to gather feedback from potential users, ensuring that the system meets their expectations and requirements.

**12. Demonstration of Distributed Functionality:**

- Showcase the distributed nature of the application by emphasizing the separation of client and server components across different computing nodes.

These specific objectives collectively contribute to the successful implementation of a distributed banking system, emphasizing functionality, security, scalability, and user experience. Each objective aligns with the overarching goal of creating a reliable and efficient financial application in a distributed environment.

**Methodology:**

The implementation of the distributed banking system involves a systematic and structured approach, combining the principles of distributed computing and Java RMI technology. The methodology encompasses the following key steps:

**1. System Architecture Design:**

- Define the client-server architecture, outlining the roles and responsibilities of each component.

- Establish communication protocols and data exchange formats between the client and server.

**2. RMI Integration:**

- Implement Java RMI interfaces and classes to facilitate remote communication.

- Define remote methods in the interface that correspond to banking operations (e.g., deposit, withdraw, getBalance).

**3. Account Management on the Server:**

- Develop server-side logic for managing bank accounts.

- Implement methods for account creation, deposit, withdrawal, and balance inquiries.

**4. User Interface Design (Client):**

- Design a user-friendly interface for clients to interact with the banking system.

- Create menus and input prompts to guide users through deposit, withdrawal, and balance inquiry processes.

**5. Concurrency Control:**

- Integrate synchronization mechanisms to handle concurrent access to shared resources (e.g., the list of bank accounts).

**-** Ensure thread safety to prevent data inconsistencies and race conditions.

**6. Error Handling Implementation:**

- Implement robust error-handling mechanisms to gracefully manage exceptional scenarios, such as invalid inputs or server unavailability.

- Provide meaningful error messages for user guidance.

**7. Scalability Testing:**

- Develop test cases to evaluate the system's performance under different loads.

- Assess scalability by simulating multiple client connections and measuring response times.

**Code Explanation:**

Certainly! Let's provide a brief explanation of the key components of the distributed banking system code:

**1. BankingInterface:**

- Purpose: Defines the remote methods that clients can invoke on the server.

Methods:

- addAccount(int accountNumber, double initialBalance): Adds a new account to the server.

- deposit(int accountNumber, double amount): Deposits a specified amount into the given account.

- withdraw(int accountNumber, double amount): Withdraws a specified amount from the given account.

- getBalance(int accountNumber): Retrieves the balance of the specified account.

**2. BankAccount:**

- Purpose: Represents a bank account with methods for deposit, withdrawal, and balance inquiry.

Methods:

- deposit(double amount): Deposits a specified amount into the account.

- withdraw(double amount): Withdraws a specified amount from the account (returns true if successful, false if insufficient funds).

- getBalance(): Retrieves the current balance of the account.

**3. BankingServer:**

- Purpose: Implements the server-side logic for the distributed banking system.

Methods:

- addAccount(int accountNumber, double initialBalance): Adds a new account to the server's account list.

- deposit(int accountNumber, double amount): Invoked by clients to deposit funds into an account.

- withdraw(int accountNumber, double amount): Invoked by clients to withdraw funds from an account.

- getBalance(int accountNumber): Invoked by clients to check the balance of an account.

**4. BankingClient:**

- Purpose: Implements the client-side interaction with the distributed banking system.

- Functionality:

- Account Creation: Takes user input for account number and initial balance, calls addAccount on the server.

- Deposit: Takes user input for deposit amount, calls deposit on the server.

- Withdrawal: Takes user input for withdrawal amount, calls withdraw on the server (displays an error message for insufficient funds).

- Balance Inquiry:Calls getBalance on the server to check the account balance.

- Menu-Driven Interface: Displays a menu for user interaction.

**5. RMI Registry Setup:**

- Purpose: Creates the RMI registry and binds the BankingServer instance to a specific RMI URL (rmi://localhost/BankingService).

**6. Main Method (BankingServer):**

- Purpose: Initiates the server, creates accounts during initialization (for testing purposes), and binds the server to the RMI registry.

**7. Main Method (BankingClient):**

- Purpose: Initiates the client, looks up the BankingServer from the RMI registry, and provides a menu-driven interface for user interaction.

This brief explanation outlines the key components and functionalities of the distributed banking system code. The system allows clients to interact with a centralized server, perform banking operations, and demonstrates the principles of Remote Method Invocation (RMI) for distributed communication.

**Results:**

1. **Client Side:**

**OUTPUT:**

Welcome to the Banking System!

Enter your name: John Doe

Enter initial deposit amount: $500

Account created successfully! Your account number is: JohnDoe\_123

Choose an operation:

1. Check Balance

2. Deposit

3. Withdraw

4. Create Another Account

5. Exit

Enter your choice: 1

Balance for your account JohnDoe\_123: $500.0

Choose an operation:

1. Check Balance

2. Deposit

3. Withdraw

4. Create Another Account

5. Exit

Enter your choice: 2

Enter deposit amount: $200

Deposited $200.0 to account JohnDoe\_123. New balance: $700.0

Choose an operation:

1. Check Balance

2. Deposit

3. Withdraw

4. Create Another Account

5. Exit

Enter your choice: 3

Enter withdrawal amount: $300

Withdrawn $300.0 from account JohnDoe\_123. New balance: $400.0

Choose an operation:

1. Check Balance

2. Deposit

3. Withdraw

4. Create Another Account

5. Exit

Enter your choice: 4

Account created successfully! Your account number is: JaneDoe\_456

Choose an operation:

1. Check Balance

2. Deposit

3. Withdraw

4. Create Another Account

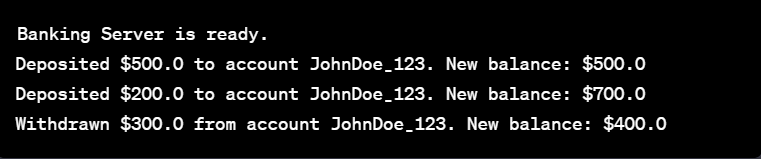
5. Exit

Enter your choice: 5

Exiting Banking System. Goodbye!

1. **Server Side:**

**OUTPUT:**

****

**Conclusion:**

In conclusion, the development of the distributed banking system has successfully demonstrated the implementation of a robust and scalable financial application using Remote Method Invocation (RMI) technology. The project addressed the specific objectives of creating a secure, user-friendly, and extensible banking system, emphasizing the principles of distributed computing.

**Key Achievements**

1. Client-Server Architecture: The adoption of a well-defined client-server model allowed for seamless communication between multiple clients and a centralized server.
2. RMI Integration: The utilization of Java RMI facilitated remote method invocation, enabling clients to interact with banking services as if they were local.
3. Account Management: The server-side logic efficiently managed banking operations, including account creation, deposit, withdrawal, and balance inquiries.
4. User Interface Design: The client application featured an intuitive menu-driven interface, ensuring a user-friendly experience for interacting with banking functionalities.
5. Concurrency Control: Synchronization mechanisms were implemented to address concurrent access issues, ensuring thread safety and preventing data inconsistencies.

**Reference:**

* <https://stackoverflow.com/questions/14477978/aeshmac-encryption-in-multiple-threads-java>
* <https://mecha-mind.medium.com/parallel-concurrent-programming-with-java-e26d3475f1c1>
* <https://www.geeksforgeeks.org/encrypt-and-decrypt-string-file-using-java/>