SOFTWARE PROJECT MANAGEMENT

[CSE432]

PRACTICAL LAB FILE

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Technology

Department of Computer Science and Engineering

Amity University, Uttar Pradesh

Submitted by: Khushi Khurana

Enrolment No.: A2305221434

Class: 7CSE4Y

Submitted to: Dr. Rajni Sehgal Kaushik

ASET

INDEX

S.No	Program	Date	Sign
1.	To design the Software Requirement Specification (SRS) document for a project	24/07/2024	
2.	To design a Software Design Document for a project	07/08/2024	
3.	To elucidate the activity schedule of the project	11/09/2024	
4.	To draw network graph and find the critical path using draw.io (CPM)	25/09/2024	
5.	To draw the Gantt chart on ClickUp software	25/09/2024	
6.	To design the PERT chart for the project	02/10/2024	
7.	To design the software plan for the project	09/10/2024	
8.	To prepare the workflow of a project	09/10/2024	
9.	To make SLIP chart, Timeline chart and ball chart	16/10/2024	

EXPERIMENT 1

AIM: To design the Software Requirements Specification document for a project.

THEORY:

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfil the needs of all stakeholders (business, users).

The elements that comprise an SRS can be simply summarized into four Ds:

- Define your product's purpose.
- Describe what you're building.
- Detail the requirements.
- Deliver it for approval.

NEED

1. Clear Communication: Ensures clear communication of project requirements among stakeholders.
2. Scope Definition: Defines project boundaries and functionalities, preventing scope creep.
3. Basis for Estimations: Provides a basis for estimating project costs, timelines, and resource requirements.
4. Quality Assurance: Forms the foundation for quality assurance by specifying expected software behavior.
5. Change Control: Facilitates evaluation of requested changes' impact on the project.
6. Risk Management: Identifies potential risks and uncertainties early in the project, allowing for proactive risk mitigation strategies.
7. Regulatory Compliance: Ensures that the software aligns with legal and regulatory requirements, which is crucial in industries with strict compliance standards.

SOFTWARE REQUIREMENT SPECIFICATIONS DOCUMENT

SOFTWARE NAME: Client Server SSH Connection

GENERAL DESCRIPTION OF THE SOFTWARE:

The software is developed in Python using Visual Studio Code and is designed to run on the Windows 11 operating system. It requires an internet connection to function properly. The development environment consists of an Intel Core i3 processor of the 10th generation.

The main functionality of the software is to read a text file containing information about remote SSH servers, including their hostnames, usernames, and passwords. It then determines the number of servers listed in the file. To efficiently handle multiple connections, the software utilizes multithreading, allowing it to connect to all the servers simultaneously. For establishing SSH connections, the Paramiko library is employed.

Once the connections are established, the software enables users to securely access the remote servers. Users can input commands like "pwd" or "ls" through the software's interface, which are then sent to the remote servers. The servers process these commands and return the output in an encrypted format. The software is capable of decrypting the output and displaying it to the user in a readable form on its interface.

In summary, the software is a multi-threaded SSH client, providing a secure way for users to remotely access multiple servers simultaneously and execute commands on them through an encrypted connection.

Table of Contents

1. INTRODUCTION

i) Purpose:

The purpose of this document is to outline the requirements and specifications for the development of a software application that aims to facilitate remote access to SSH servers by securely logging into them and executing commands. The software is written in Python, using Visual Studio Code as the integrated development environment (IDE).

ii) Document Conventions:

Title Formatting:

- Main section titles: All main section titles (e.g., Introduction, Overall Description, External Interface Requirements) are in bold, capitalized and numbered.
- Subsection titles: Subsection titles (e.g., Purpose, Product Functions, User Interfaces) are in bold and are in roman number.
- Subsubsection titles: Subsubsections (if used) are in italics and in bullets.
- Sub-sectional points: Subsubsections (if used) are in plain format and are in circle bullet.
- Normal text: The main content of the document is written in plain, black font and in Times New Roman.

Lists:

- Bullet points: Lists with non-sequential items (e.g., product features, assumptions) use bullet points.
- Numbered lists: Lists with sequential items (e.g., steps of a process) use numbered lists.

References and Citations:

- References to external sources, such as libraries or specifications, are appropriately cited using standard citation formats.

Images and Diagrams:

- Images and diagrams are used to enhance understanding and are labelled with appropriate captions and figure numbers.

Page Layout and Pagination:

- Standard page layout and pagination are followed, including page numbers and headers/footers with relevant information.

iii) Intended Audience and Reading Suggestions:

The intended audience for this software documentation includes the following types of readers:

- *Developers:* Developers who will be involved in maintaining, enhancing, or troubleshooting the software. They will be interested in understanding the technical aspects of the software, its architecture, and implementation details.
- *Users:* Users who will interact with the software to remotely access and manage the SSH servers. They will be interested in understanding how to use the software effectively, including the user interface, available commands, and security practices.
- *Testers:* Testers responsible for testing the software's functionality, performance, and security. They will be interested in understanding the testing requirements, test cases, and expected behaviour.

Readers are suggested to start with the overview sections to gain an understanding of the software's context, purpose, and constraints. Then, proceed to the detailed sections, such as System to understand the software's functionalities, and External Interface Requirements to understand the user interfaces and software dependencies.

iv) Project Scope:

The scope of this project includes the development of a software application on a client machine that can read a text file containing SSH server information, establish simultaneous connections to the listed servers using multithreading, and provide a secure interface for the user to execute commands to the remote servers and view their output.

v) Overview:

This section provides a brief overview of the software, highlighting its main functionalities and objectives. The software utilizes the Paramiko Python library for SSH connections and operates on the Windows 11 operating system. Internet connectivity is required for proper functioning.

2. OVERALL DESCRIPTION

i) Product Perspective:

The SSH server management software is designed to operate as a standalone application. It interacts with remote SSH servers based on the information provided in the input text file, establishing secure connections and facilitating command execution on multiple servers concurrently.

ii) Product Functions

The software performs the following key functions:

- Server Information Reading: Parses the input text file to retrieve hostname, username, and password details of remote SSH servers.
- Multithreaded SSH Connections: Establishes concurrent connections to multiple remote servers using multithreading for improved efficiency.
- Secure Remote Access: Provides secure login to the remote servers through encrypted communication channels.
- Remote Command Execution: Accepts user commands (e.g., pwd, ls) and sends them to the remote servers for execution.
- Encrypted Output Decryption: Decrypts the output received from remote servers in encrypted format for display to the user.

The software performs the following tasks in detail:

- Reads a text file containing hostname, username, and password information of remote SSH servers.
- Counts the number of servers listed in the text file.
- Establishes simultaneous connections to all listed servers using multithreading.
- Provides a secure interface for the user to input commands such as "pwd" and "ls".
- Sends the user commands to the remote servers for execution.
- Receives the encrypted output from the remote servers.
- Decrypts the output and displays it to the user on the software interface.

iii) User Classes and Characteristics

The software targets the following user classes:

- Developers: Familiar with Python programming and network concepts.
- Users: May have basic knowledge of SSH and command-line interfaces.

- Testers: Proficient in software testing methodologies and SSH server testing.

iv) Operating Environment

The software is developed for the Windows 11 operating system and requires internet connectivity to establish SSH connections to remote servers.

v) Design and Implementation Constraints

- Multithreading: The software leverages multithreading for simultaneous server connections, considering system resource limitations.
- Security: Strong encryption mechanisms are employed to protect sensitive information during SSH connections.
- Python and Paramiko Library: The software is implemented using Python, and the Paramiko library is used for SSH connections.

vi) User Documentation

Comprehensive user documentation, including user guides and manuals, is provided to assist users in understanding how to interact with the software effectively.

vii) Assumptions and Dependencies

The input text file contains correct and valid server information (hostname, username, password) for successful connections.
The remote SSH servers support SSH connections and allow the specified username and password for login.

3. EXTERNAL INTERFACE REQUIREMENTS

i) User Interfaces

The user interface of the software provides a straightforward and intuitive experience, allowing users to securely access multiple remote SSH servers simultaneously. The interface displays server information from a text file and offers standard buttons like "Connect," "Help," and "Disconnect." Users can input commands in a command prompt-like area, receiving encrypted server outputs, which are decrypted and displayed on the interface for efficient server management. The design adheres to GUI standards, platform-specific guidelines, and includes error message displays for a seamless and user-friendly experience.

ii) Hardware Interfaces

The software, developed on Windows 11 OS and written in Python using Visual Studio Code, interacts with the hardware components of the system in a seamless manner. It is compatible with devices equipped with an Intel Core i3 processor of the 10th generation or higher. The software requires internet connectivity to function properly. It establishes logical interfaces with the hardware through standard communication protocols to facilitate reading of the text file containing remote server information. The software utilizes multithreading to efficiently connect to multiple servers simultaneously. Through the Paramiko Python library, the software establishes secure SSH connections to the remote servers, enabling encrypted data exchange between the software and the hardware. The software's physical characteristics ensure smooth execution, secure data transfer, and reliable remote access to the servers, enhancing user experience and convenience.

iii) Software Interfaces

The software connects with various components to function seamlessly, including interacting with Windows 11 to access the file system and read a text file with remote server info. For SSH connections, it uses Paramiko Python library to log into servers securely. Multithreading enables efficient connections to multiple servers simultaneously for improved performance. Data items like hostname, username, and password are exchanged to establish connections and execute commands like "pwd" and "ls". Encrypted output from the servers is decrypted and presented on the software interface for remote server access and management. Specific protocols ensure smooth communication between software components.

iv) Communication Interfaces

The software relies on internet connectivity for its communication functions, facilitating seamless remote access to the remote SSH servers. It uses the Paramiko Python library to establish secure SSH connections, ensuring encrypted data exchange between the software and the remote servers. The communication occurs over standard network server communications protocols like TCP/IP. The software is designed to efficiently handle multithreading, enabling simultaneous connections to multiple servers, which enhances data transfer rates and synchronization mechanisms. As a security measure, the software encrypts the command output received from the remote servers and decrypts it for display on the software interface. Communication interfaces do not involve email or web browser functionalities, focusing solely on secure and efficient SSH communication for remote server management.

4. SYSTEM FEATURES

i) System Feature 1: Text File Parsing and Server Enumeration

The software begins by reading a text file containing critical information about multiple remote SSH servers, such as their hostnames, usernames, and passwords. This feature streamlines the process of managing multiple servers by centralizing the necessary credentials in a structured format. The software intelligently parses the text file, extracting the server details and creating a list of servers to be accessed. This user-friendly approach saves time and reduces potential errors in manual input, ensuring a smooth and organized workflow for remote server management.

ii) System Feature 2: Multithreaded Concurrent Connections

Utilizing the power of Python's multithreading capabilities, the software efficiently establishes concurrent connections to all the remote servers enlisted in the text file. By employing the capabilities of the 10th generation Intel Core i3 processor, the software maximizes resource utilization and minimizes connection time, significantly enhancing overall performance. This threading feature allows users to access multiple servers simultaneously, drastically reducing the time required for remote server administration and making the software a versatile tool for handling large-scale server infrastructures.

iii) System Feature 3: Secure SSH Connection using Paramiko

To ensure secure and encrypted communication with remote servers, the software relies on the Paramiko Python library for SSH connections. This feature guarantees the confidentiality and integrity of the data transmitted between the user and the remote servers. By leveraging Paramiko's robust encryption and authentication mechanisms, the software safeguards sensitive information, such as login credentials and command outputs, from potential eavesdropping and tampering attempts. This security-centric approach fosters a trustworthy

and reliable environment for remote access, making the software a preferred choice for maintaining the privacy of critical server interactions.

iv) System Feature 4: Encrypted Command Execution and Output Decryption

The software acts as an intermediary between the user and the remote servers, securely sending user commands (e.g., pwd, ls) to the servers for execution. The output generated by these commands is received in an encrypted format and subsequently decrypted by the software before being displayed to the user on the software interface. This feature ensures that sensitive information exchanged during the remote sessions remains confidential throughout the process. By handling encryption and decryption seamlessly, the software guarantees a seamless and secure user experience while managing remote servers effectively.

5. OTHER NONFUNCTIONAL REQUIREMENTS

i) Performance Requirements:

To meet the performance requirements, this software should aim for efficient resource utilization and responsiveness during its execution. Given that the software is designed to handle multiple SSH connections simultaneously, it should be optimized to make the best use of the underlying hardware, including the 10th generation Intel Core i3 processor, to achieve smooth multithreading and concurrent operations. Ensuring low latency in connecting to the remote servers is essential to reduce user wait times. The software should be capable of handling a large number of servers listed in the text file without significant performance degradation. Efficient memory management is crucial to avoid potential bottlenecks and crashes during prolonged usage. Additionally, the Paramiko library should be well-configured to maintain secure and reliable SSH connections with minimal overhead. Regular updates and performance profiling will help identify and address any potential performance bottlenecks, enhancing the software's overall stability and responsiveness during remote server management tasks.

ii) Safety Requirements:

The safety requirements for this software are paramount to ensure the secure management of remote servers and safeguard sensitive information. Implementing strong data encryption and secure communication protocols protects against unauthorized access and eavesdropping. Robust authentication and authorization mechanisms ensure that only authorized users can access the software and the corresponding servers. Secure password handling using hashing and encryption prevents plaintext storage of credentials. Proper validation of user input and comprehensive error handling protect against injection attacks and prevent crashes. Enabling audit logs allows for the monitoring of critical events and suspicious activities. Regular software updates, restricted access, and security testing further enhance the software's resilience to potential vulnerabilities. Providing clear documentation and training for users promotes security awareness and best practices, ensuring a safe and reliable environment for remote server management.

iii) Security Requirements:

The security requirements for this software are crucial to establish a robust and trustworthy system for managing remote servers securely. First and foremost, the software must implement strong authentication mechanisms to verify the identity of users before granting access to the remote servers. This includes multi-factor authentication options for an added

layer of protection. All sensitive data, including login credentials and command outputs, must be encrypted during transmission using industry-standard encryption protocols like RSA/DES. Access controls should be enforced diligently to restrict user privileges and ensure that each user can only interact with authorized servers and perform permitted actions. To prevent unauthorized access and potential breaches, the software should undergo regular security audits, including penetration testing, to identify and address vulnerabilities proactively. It is essential to stay up-to-date with security patches and updates for the software and its dependencies to mitigate newly discovered threats. User activity and system events should be logged thoroughly, enabling monitoring and detection of any suspicious behavior. Finally, a clear security policy and user training should be in place to educate users on security best practices and create a security-conscious culture surrounding the software's usage. By adhering to these security requirements, the software can establish a reliable and protected environment for remote server management.

iv) Software Quality Attributes:

The software quality attributes of this application play a vital role in ensuring its overall effectiveness, usability, and maintainability. First and foremost, the software should exhibit high reliability, ensuring that it performs consistently and accurately during remote server management operations. Reliability is crucial to avoid downtime and potential data loss, particularly when dealing with critical server infrastructures. Additionally, the software should prioritize performance optimization, efficiently utilizing system resources and minimizing latency during multithreaded connections to provide a seamless and responsive user experience. Usability is another important attribute, requiring an intuitive and user-friendly interface that simplifies server enumeration, command execution, and access management. Software maintainability is essential for long-term viability, necessitating well-structured and modular code, allowing for easy updates, bug fixes, and improvements. Lastly, the software should adhere to stringent security measures, as discussed earlier, to ensure data confidentiality, integrity, and protection against potential threats. By focusing on these software quality attributes, the application can deliver a reliable, high-performance, user-friendly, and secure platform for effectively managing remote servers.

v) Business Rules:

The software, designed for remote server management, relies on well-defined business rules to ensure its alignment with the organization's specific needs and objectives. Authentication and authorization policies govern user access, roles, and permissions, safeguarding sensitive functionalities and data. Access restrictions are established to define who can access specific servers and under what conditions, enhancing security. Data retention policies manage storage requirements and comply with data protection regulations. Logging and auditing rules track critical events, while error handling and reporting mechanisms ensure graceful recovery from issues. Concurrency limits optimize resource management, and user account rules define registration, modification, and termination procedures. Compliance with regulatory standards is addressed, along with strong encryption and data security measures. Additionally, software licensing and usage rules ensure adherence to licensing agreements, making the software a robust, compliant, and secure solution for remote server management.

6. OTHER REQUIREMENTS

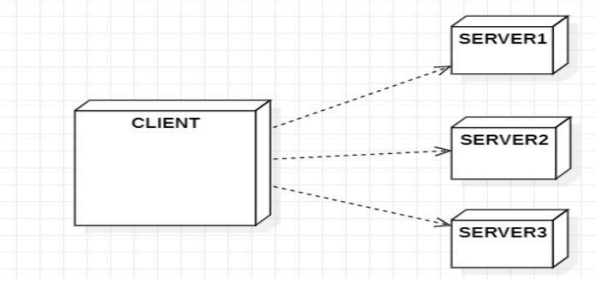
i) Appendix A: Glossary

1. SSH protocol: SSH (Secure Shell) is a network protocol that provides secure and encrypted remote access and management of systems over potentially unsecured networks. It replaces older, less secure protocols like Telnet and rlogin. SSH allows for secure authentication, including password-based and public key-based methods. Once connected, it establishes an encrypted tunnel for secure data transmission between the client and server. SSH is widely used for remote administration and secure communication, making it a crucial component of modern cybersecurity practices.

2. Paramiko library: The Paramiko library is a powerful and widely-used Python module for handling SSH (Secure Shell) connections. It provides an easy-to-use interface to establish secure communication with remote servers, making it ideal for automating tasks, remote management, and secure data transfer. Paramiko enables users to create SSH clients and servers, execute commands remotely, and transfer files securely. Its support for various encryption algorithms, authentication methods, and advanced features like tunneling makes it a preferred choice for developers working on SSH-based applications and systems. Whether for simple scripting or complex network automation, Paramiko simplifies SSH communication and enhances the security of remote interactions.

ii) Appendix B: Analysis Model

The following diagram shows a single client connects with multiple servers to send commands and receive outputs.



EXPERIMENT 2

AIM: To design a Software Design Document for the project Client Server SSH Connection.

THEORY: Software design documents help to ensure the design specifications of the software are understood and it's clear to all what is possible and how it can be accomplished. They are an important way of looping everyone in the team into the process who is involved in the product. The basic format or template for a software design document is:

1. Title
2. Introduction: Provide an overview of the entire document
3. System Overview: Provide a general description and functionality of the software system.
4. Design Considerations: Describe the issues that need to be addressed before creating a design solution
5. Assumptions and Dependencies: Describe any assumptions that may be wrong or any dependencies on other things.
6. General Constraints: Describe any constraints that could have an impact on the design of the software.
7. Goals and Guidelines: Describe any goals and guidelines for the design of the software.
8. Development Methods: Describe the software design method that will be used.
9. Architectural Strategies: Describe the strategies used that will affect the system.
10. System Architecture: This section should provide a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components.
11. Policies and Tactics: Describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures).
12. Detailed System Design: Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well.
13. Glossary: An ordered list of defined terms and concepts used throughout the document.

RESULT:

SOFTWARE DESIGN DOCUMENT

SOFTWARE NAME: Client Server SSH Connection

GENERAL DESCRIPTION OF THE SOFTWARE:

The software is developed in Python using Visual Studio Code and is designed to run on the Windows 11 operating system. It requires an internet connection to function properly. The development environment consists of an Intel Core i3 processor of the 10th generation.
The main functionality of the software is to read a text file containing information about remote SSH servers, including their hostnames, usernames, and passwords. It then determines the number of servers listed in the file. To efficiently handle multiple connections, the software utilizes multithreading, allowing it to connect to all the servers simultaneously. For establishing SSH connections, the Paramiko library is employed.
Once the connections are established, the software enables users to securely access the remote servers. Users can input commands like "pwd" or "ls" through the software's interface, which are then sent to the remote servers. The servers process these commands and return the output in an encrypted format. The software is capable of decrypting the output and displaying it to the user in a readable form on its interface.
In summary, the software is a multi-threaded SSH client, providing a secure way for users to remotely access multiple servers simultaneously and execute commands on them through an encrypted connection.

Table of Contents

CLIENT SERVER SSH CONNECTION

1. INTRODUCTION

This document outlines the design of the “Client Server SSH Connection” software. It provides an overview of its purpose, architecture, functionality, design considerations, assumptions, dependencies and constraints. It also details the development methods, system architecture, policies and tactics followed along with the technical design of the individual modules involved.

2. SYSTEM OVERVIEW

The software project is a multi-threaded SSH client written in Python. It reads server information from a text file and establishes SSH connections to multiple remote servers using Paramiko, a library that facilitates secure communication. The key functionality includes executing user commands like ‘pwd’ or ‘ls’ on all connected servers in parallel and displaying the encrypted output. The system is designed to run on Windows 11 and requires an internet connection for operation.

Key Features:

- File-based input for server credentials
- Parallel SSH connections using multithreading
- Command execution with encrypted communication
- Readable output of remote server responses

3. DESIGN CONSIDERATIONS

- Performance: The system must manage multiple server connections efficiently.
- Security: Secure communication through SSH to prevent data breaches.
- Usability: Ensure simple command input and clear output for the user.
- Scalability: Ability to handle an increasing number of server connections.
- Error Handling: Must gracefully manage connection failures or incorrect login details.

4. ASSUMPTIONS AND DEPENDENCIES

- **Assumptions:**
    - All servers listed in the input file are configured for SSH.

        - The provided usernames and passwords are correct.
    - **Dependencies:**
        - Paramiko library is needed for SSH functionality.
        - Python 3.x must be installed for code execution.
        - Active internet connection is required to connect to remote servers.

5. GENERAL CONSTRAINTS

- **Operating system:** Designed for Windows 11
- **Hardware:** Minimum requirement of an Intel Core i3, 10th Gen processor
- **Library Support:** Paramiko and threading libraries must be available
- **Network Stability:** Unstable connections may cause disruptions

6. GOALS AND GUIDELINES

- **Goals:**
    - Provide a secure and efficient way to access multiple servers remotely.
    - Support seamless parallel command execution.
- **Guidelines:**
    - Use Python's threading features to manage multiple SSH connections.
    - Apply proper error handling for failed connections and commands.
    - Follow standard practices for secure data communication.

7. DEVELOPMENT METHODS

This software is developed using an incremental development process, where each core component (file reading, SSH connection and command execution) is built and tested individually. Integration and testing occur iteratively to ensure all modules work together without issues.

8. ARCHITECTURAL STRATEGIES

- **Client server model:** The software acts as a client that connects to multiple servers simultaneously.
- **Multithreading:** Utilized to connect to multiple servers concurrently without blocking other operations.

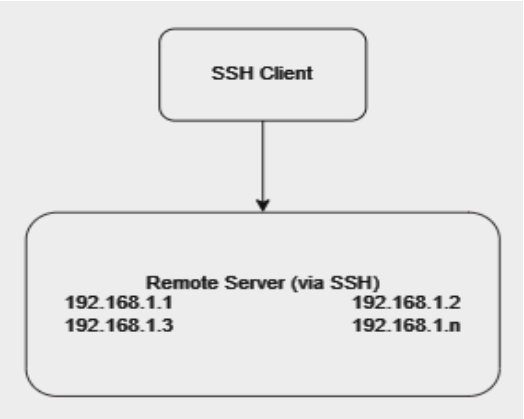
- **CLI interface:** Commands are provided through a terminal and results are displayed as plain text.

9. SYSTEM ARCHITECTURE

The system consists of the following modules:

- **File Reader Module:** Reads the server details (hostname, username, password) from a text file.
- **Connection Manager Module:** Establishes and maintains SSH connections using Paramiko and handles multiple threads.
- **Command Executor Module:** Takes user commands and sends them to the connected servers. Retrieves the output for display.
- **Decryption and Output Display Module:** If needed, decrypts the response and presents it to the user in an understandable format.

10. ARCHITECTURE OVERVIEW



11. POLICIES AND TACTICS

- **Error Logging:** Log failed connections and commands for troubleshooting purposes.
- **Resource Management:** Monitor and limit the number of threads to prevent system overload.
- **Security Measures:** Use SSH for encrypted communication and ensure no sensitive data is exposed.

12. DETAILED SYSTEM DESIGN

1. **File Reader Module:**

   Function: Reads the list of servers from a text file

Implementation:

```
def read_servers(file_path):
    servers = []
    with open(file_path, 'r') as file:
        for line in file:
            host, user, password = line.strip().split(',')
            servers.append((host, user, password))
    return servers
```

2. **Connection Manager Module:**

   Function: Establishes SSH connections using Paramiko and manages threads for simultaneous connections.

   Implementation:

```
import paramiko
import threading
def connect_to_server(host, user, password):
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=host, username=user, password=password)
        print(f"Connected to {host}")
        return ssh
    except Exception as e:
        print(f"Failed to connect to {host}: {e}")

def connect_all_servers(servers):
    threads = []
    for host, user, password in servers:
            thread = threading.Thread(target=connect_to_server, args=(host, user, password))
        thread.start()
        threads.append(thread)
    for thread in threads:        thread.join()
```

3. **Command Executor Module:**

Function: Executes user commands on the connected servers and retrieves their output.

Implementation:

```
def execute_command(ssh, command):
    stdin, stdout, stderr = ssh.exec_command(command)
    return stdout.read().decode('utf-8')
```

4. **Decryption and Output Display Module:**

   Function: If needed, decrypts the output and displays it in readable form.

13.GLOSSARY

- **SSH (Secure Shell):** A protocol for securely accessing remote systems over a network.
- **Paramiko:** A Python library for establishing SSH connections.
- **Multithreading:** A programming technique that allows multiple tasks to run concurrently.
- **CLI (Command-Line Interface):** A text-based interface where users interact with the system using commands.
- **Encryption:** The process of converting data into a secure format to prevent unauthorized access.
- **Decryption:** The process of converting encrypted data back into its original form.

EXPERIMENT 3

AIM: To elucidate the activity schedule of the project

THEORY: In software project management, an activity refers to a specific task or piece of work that needs to be completed as part of the project's overall plan. They are typically defined, scheduled, assigned to team members and tracked to ensure they are completed on time and within the project's constraints. The time spans used for constructing a PERT chart are given below:

1. Duration of the activity: the time it takes to complete a specific task or activity.
2. Earliest start: the earliest point in time when a task can begin based on its dependencies and the project schedule.
3. Earliest finish: The earliest point in time when a task can be completed, considering its earliest start and duration.
4. Latest start: The latest allowable point in time when a task can start without delaying the project's completion date.
5. Latest finish: The latest allowable point in time when a task can be completed without delaying the project's completion date.
6. Float: The amount of time a task can be delayed without affecting the project's overall duration, also known as slack time.

RESULT:

ACTIVITY LABEL	DURATION	EARLIEST START	EARLIEST FINISH	LATEST START	LATEST FINISH	FLOAT
Requirement and Feasibility Analysis	4	1	4	1	4	0
SRS Design	3	4	7	4	7	0
Literature Review	2	7	9	7	9	0
Frontend code	1	9	10	10	11	0
Backend code	2	9	11	9	11	0
Testing	2	11	13	11	13	0
Deployment	2	13	15	13	15	0
Report Writing	3	15	18	15	18	0

EXPERIMENT 4

AIM: To draw network graph and find the critical path using draw.io (CPM)

THEORY: A network diagram in project management is a visual representation of project tasks and their relationships, illustrating the sequence of activities, dependencies, and critical path to help plan and manage project execution effectively. By mapping out these relationships, project managers can identify the sequence in which tasks need to be completed and determine the critical path—the longest sequence of dependent tasks that determines the project's duration. Network diagrams help project managers:

- Sequence Tasks: They provide a clear understanding of the order in which tasks should be executed, ensuring that work progresses smoothly from one activity to the next.
- Identify Dependencies: By showing the relationships between tasks, network diagrams highlight dependencies—where the completion of one task depends on the start or completion of another. This helps in managing task dependencies effectively.
- Determine the Critical Path: The critical path is the longest sequence of tasks that determines the shortest possible duration of the project. By identifying the critical path, project managers can focus their efforts on managing and optimizing tasks along this path to ensure timely project completion.
- Estimate Project Duration: By analyzing the sequence of tasks and dependencies, project managers can estimate the overall duration of the project more accurately, taking into account potential delays and dependencies.
- Allocate Resources: Network diagrams help in resource allocation by providing insights into when and where resources are needed throughout the project lifecycle.

Overall, network diagrams are valuable tools for project managers to plan, execute, and monitor projects effectively, ensuring that tasks are completed in the right order and project objectives are achieved within the desired timeframe.

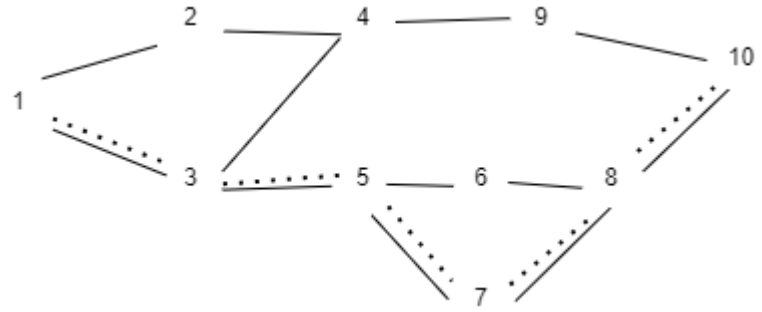
Given the activity schedule of a project:

ACTIVITY	TIME (IN DAYS)
1-2	4

1-3	1
2-4	1
3-4	1
3-5	6
4-9	5
5-6	4
5-7	8
6-8	1
7-8	2
8-10	5
9-10	7

RESULT:

The figure below shows the network diagram of the above activity schedule and the dotted lines depict the critical path.



Critical path: 1-3-5-7-8-10

EXPERIMENT 5

AIM: To draw the Gantt chart on ClickUp software

THEORY:

A Gantt chart is a horizontal bar chart used in project management to visually represent a project over time. Gantt charts typically show you the timeline and status as well as who's responsible for each task in the project. In project management, Gantt charts are used to schedule, track, and communicate deliverables, deadlines, dependencies, and resource assignments. They are particularly helpful in managing complex projects with interdependencies that a simple to-do list or Kanban board can't handle. A Gantt chart allows you to simplify complex projects into an easy-to-follow plan that includes:

- How a project breaks down into tasks
- When each task will begin and end
- How long each task will take
- Who's assigned to each task
- How tasks relate to and depend on each other
- When important meetings, approvals, or deadlines need to happen
- How work is progressing in a project
- The full project schedule from start to finish

The basic parts of a Gantt chart to understand how each element functions in a project plan are as follows:

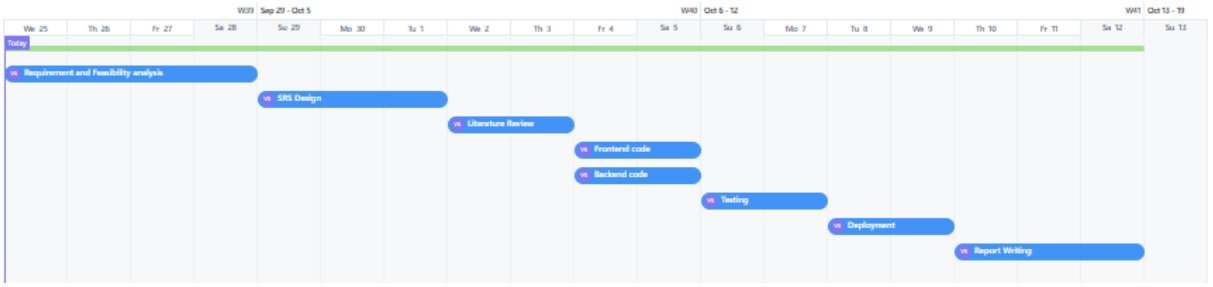
- Task list: Runs vertically down the left of the gantt chart to describe project work and may be organized into groups and subgroups
- Timeline: Runs horizontally across the top of the gantt chart and shows months, weeks, days, and years
- Dateline: A vertical line that highlights the current date on the gantt chart
- Bars: Horizontal markers on the right side of the gantt chart that represent tasks and show progress, duration, and start and end dates
- Milestones: Yellow diamonds that call out major events, dates, decisions, and deliverables
- Dependencies: Light gray lines that connect tasks that need to happen in a certain order
- Progress: Shows how far along work is and may be indicated by percent complete and/or bar shading

- Resource assigned: Indicates the person or team responsible for completing a task

Given the following project phases and their respective duration:

ACTIVITY	TIME (IN DAYS)
Requirement and Feasibility Analysis	4
SRS Design	3
Literature Review	2
Frontend code	1
Backend code	2
Testing	2
Deployment	2
Report Writing	3

RESULT:



EXPERIMENT 6

AIM: To draw the PERT chart for a project

THEORY:

A PERT chart is a visual project management tool used to map out and track the tasks and timelines. The name PERT is an acronym for Project (or Program) Evaluation and Review Technique. PERT charts can be drawn as free-form diagrams. Project managers create PERT charts by drawing boxes or nodes representing events and connecting them via arrows, representing the tasks that must be completed between each milestone and the amount of time the team will have to complete each task.

The components of a typical PERT chart are explained as follows:

1. Numbered nodes: Each node represents an event or milestone in the project, completion of one stage, or a series of tasks needed to move the project forward.
2. Directional (or concurrent) arrows: The arrows on a PERT chart represent the tasks or activities that need to be completed before the team can move on to the next event or phase in the project. Project managers use directional arrows to schedule activities that have dependencies.
3. Divergent arrows: These arrows represent tasks that a team may work on simultaneously or in any sequence they choose because they do not have dependencies.

To create a PERT chart, a project management team should follow these steps.

Step 1: Identify all of the project's activities: First, define all of the major phases, milestones, and tasks needed to complete the project.

Step 2: Identify dependencies: If you determine some tasks or activities have dependencies, you will want to depict those tasks with directional arrows. This will ensure your team knows the sequence they need to tackle each task.

Step 3: Draw your chart: The next step is to take the identified events and milestones (numbered nodes) and draw them out. Then write out the tasks and activities that the team must complete between each node, using directional arrows or divergent arrows accordingly.

Step 4: Establish timelines for all activities: You should now set a timeframe when the team will need to complete those tasks along with all arrows.

Given the following activity schedule with the predecessors and durations:

Activity	Predecessor	t0	tm	tp

A	-	2	4	9
B	A	5	8	14
C	B	4	10	13
D	B	4	7	10
E	C	11	14	20
F	D	9	13	16
G	E, F	2	4	6

RESULT:

Activity	Predecessor	t0	tm	tp	Exp time	variance	S.D.
A	-	2	4	9	4.5	1.361	1.166
B	A	5	8	14	8.5	2.25	1.5
C	B	4	10	13	9.5	2.25	1.5
D	B	4	7	10	7	1	1
E	C	11	14	20	14.5	2.25	1.5
F	D	9	13	16	12.83	1.361	1.166
G	E, F	2	4	6	4	0.445	0.667

EXPERIMENT 7

AIM: To design the software plan for the project

THEORY:

The project plan is the controlling document to manage an Information Management/Information Technology (IM/IT) project. The project plan describes the:

• Interim and final deliverables the project will deliver,

• Managerial and technical processes necessary to develop the project deliverables,

• Resources required to deliver the project deliverables, and

• Additional plans required to support the project.

Documenting the decisions is essential. As you record, gaps appear and inconsistencies protrude. Creating the project plan usually requires hundreds of mini-decisions and these bring clarity to the project. The project plan communicates the decisions to others. Often what we assume is common knowledge is unknown by other members of the team. Fundamentally, the project manager's goal is to keep everyone progressing in the same direction and communication is essential to achieve this goal. The project plan makes communicating a lot easier. The project plan is a wealth of information as well as a checklist. By reviewing the project plan, as often as is required, the project manager knows where the project is to identify what correction action or changes of emphasis or shifts in direction are needed. 80% of a project manager's time is spent on communication: hearing, reporting, teaching, counselling, and encouraging. The other 20% is spent on activities where the project manager needs information that is data-based. The project plan is a critical set of documents that should meet this need. The job of the project manager is to develop a project plan and to accomplish it. Only the written project plan is precise and communicable. The project plan consists of documents on who, what, why, when, where, how and how much. The project plan encapsulates much of the project manager's work. If their comprehensive and critical nature is recognized in the beginning, the manager can approach them as friendly tools rather than annoying overhead. The project managers can set the direction much more crisply and quickly by doing so.

RESULT:

TABLE OF CONTENTS

1. PROJECT OVERVIEW

1.1 PURPOSE, SCOPE AND OBJECTIVES

- The purpose of this project is to develop a Python SSH software that allows users to remotely access and manage remote SSH servers. The software will provide a secure and user-friendly interface for users to interact with the remote servers.
- Scope of the software should be able to handle different types of SSH servers, including Linux and Unix servers.
- The objectives of this project are to:

❖ Develop a Python SSH software that is easy to use and provides a secure way for users to remotely access and manage remote SSH servers.
❖ Use the Paramiko Python library to securely connect to the remote SSH servers.
❖ Implement multithreading to allow the software to connect to multiple remote SSH servers simultaneously.
❖ Develop a user-friendly interface for users to interact with the remote servers.

1.2 ASSUMPTIONS AND CONSTRAINTS

The following assumptions are made about the Python SSH software project:

- Users have a basic understanding of CLI Commands like pwd.
- The remote SSH servers are running a supported version of SSH.
- The remote SSH servers are accessible over the network.
- Login credentials for the remote systems are available.
- The client side system must have SSH Port 22 open.

The following constraints apply to the Python SSH software project:

- The software must be developed using the Python programming language.
- The software must use the Paramiko Python library to securely connect to the remote SSH servers.
- The software must be able to run on Windows.
- The software must meet certain security requirements.
- The software must be scalable to support a certain number of users and remote SSH servers.
- The text file must be easy to maintain and update.

1.3 PROJECT DELIVERABLES

The following are the deliverables for the Python SSH software project:

- Working Python SSH software that allows users to remotely access and manage remote SSH servers.
- An updatable text file with credentials of all the remote servers
- Documentation for the software, including a user guide and a technical reference manual.
- Test cases and results to demonstrate that the software meets the requirements.
- A deployment plan for the software.

1.4 SCHEDULE AND BUDGET SUMMARY

The following is a high-level schedule and budget summary for the Python SSH software project:

Task	Start Date	End Date
Requirements gathering	09/20/2023	09/25/2023
Design	09/26/2023	10/07/2023
Development	10/08/2023	11/04/2023
Testing	11/05/2023	11/18/2023
Deployment	11/19/2023	11/25/2023

Expense	Cost
Software development tools	$100
Cloud hosting	$50/month
Documentation tools	$200
Testing tools	$100
Total	$750

1.5 EVOLUTION OF THE PLAN

- Risk management: Identify and assess risks, develop mitigation plans.
- Communication plan: Outline how the project team will communicate with each other and with stakeholders, including frequency and method.
- Quality assurance plan: Outline how the project team will ensure that the project deliverables meet the required quality standards, including testing and inspection procedures.
- Acceptance criteria: Develop acceptance criteria that must be met in order for the project deliverables to be accepted by the stakeholders.

1.6 REFERENCES

- A Deeper Understanding of SSH: Results from Internet-wide Scans (2014) by Oliver Gasser, Ralph Holz, and Georg Carle. This paper presents the results of a large-scale scan of the Internet for SSH servers. The authors found that many SSH servers are vulnerable to attack, and they provide recommendations for improving SSH security.
- SSH Authentication: A Comprehensive Survey (2018) by Mohammadreza Naseri, Mohammad Reza Katebzadeh, and Ali Dehghantanha. This paper surveys the different SSH authentication methods and their security implications. The authors also provide recommendations for choosing the right SSH authentication method for your needs.
- **SSH: The Secure Shell:** https://www.amazon.com/SSH-Secure-Shell-Definitive-Guide/dp/0596008953**)**
- **SSH Keys:** https://dev.to/risafj/ssh-key-authentication-for-absolute-beginners-in-plain-english-2m3f
- **SSH Tutorial:** https://m.youtube.com/watch?v=v45p_kJV9i4
- **SSH Tunneling:** https://bioteam.net/blog/tech/networking/security/ssh-tunnels-for-beginners/

1.7 DEFINTIONS AND ACRONYMS

- SSH: Secure Shell, is a network protocol that provides a secure way to access a remote computer. It encrypts all traffic between the two computers, including the user's login credentials and the commands and output of terminal sessions. This makes it a much safer alternative to other protocols, such as telnet and FTP, which transmit data in plain text.

- SFTP: SSH File Transfer Protocol
- SCP: Secure Copy
- RSA: Rivest–Shamir–Adleman is a public-key cryptosystem that is widely used for secure data communication. It is based on the mathematical difficulty of factoring large numbers. RSA works by using two different keys: a public key and a private key. The public key can be shared with anyone, but the private key must be kept secret. To encrypt a message using RSA, the sender uses the recipient's public key. The encrypted message can then only be decrypted using the recipient's private key.
- DSA: Digital Signature Algorithm
- AES: Advanced Encryption Standard
- HMAC: Hash-based message authentication code
- SSH Agent: A program that stores and manages SSH private keys
- SSH Server: A program that listens for and accepts SSH connections
- SSH Client: A program that initiates SSH connections

2. PROJECT ORGANIZATION

2.1 EXTERNAL INTERFACES

External interfaces are the boundaries between a system and its environment. They define how the system interacts with other systems and with the outside world. External interfaces can be physical, such as a network connection or a file system, or they can be logical, such as an API or a message queue. It is important to carefully design and manage external interfaces to ensure that the system is secure, reliable, and efficient.

Here are some examples of external interfaces:

- A network interface card (NIC) that connects a computer to a network.
- A file system that allows a program to access data stored on disk.
- A database interface that allows a program to access data stored in a database.
- An API that allows a program to interact with another program or service.
- A message queue that allows two or more programs to communicate with each other by sending and receiving messages.

2.2 INTERNAL STRUCTURE

The internal structure of a system is the way that the system is organized and how its components interact with each other. The internal structure of a system can be complex, but it is important to have a good understanding of it in order to develop, maintain, and troubleshoot the system.

Here are some examples of internal structure components:

- Modules: Modules are self-contained units of code that can be reused and combined to create larger programs.
- Classes: Classes are blueprints for creating objects. Objects are instances of classes and they have their own properties and methods.
- Functions: Functions are reusable blocks of code that perform specific tasks.

- Variables: Variables are used to store data.
- Data structures: Data structures are used to organize data in a way that is efficient and easy to use.

2.3 ROLES AND RESPONSIBILITIES

Roles and responsibilities are the different tasks and responsibilities that are assigned to different people or groups in a project or organization. It is important to clearly define roles and responsibilities to ensure that everyone knows what they are supposed to do and to avoid conflicts and confusion.

Here are some examples of roles and responsibilities:

- Project manager: The project manager is responsible for overseeing the project and ensuring that it is completed on time and within budget.
- Software developer: The software developer is responsible for writing and testing the code for the software.
- Quality assurance engineer: The quality assurance engineer is responsible for testing the software to ensure that it meets the requirements and that it is free of defects.
- System administrator: The system administrator is responsible for installing, configuring, and maintaining the system's hardware and software.

3. MANAGERIAL PROCESS PLANS

3.1 START-UP PLAN

3.1.1 Estimates

- Conduct a thorough analysis of the project requirements and specifications to determine the scope of the software.
- The cost of a laptop with Windows 11 OS, an Intel Core i3 processor (10th generation), and a server can vary greatly depending on the specific configurations.
- Some approximate prices based the requirements are:
- Laptop: The Infinix INBook Y1 Plus with an Intel Core i3 10th Gen 1005G1 processor, 8 GB RAM, and a 512 GB SSD running Windows 11 Home is priced around ₹38,990.
- Server: The prices for servers can range from as low as ₹5,000 to over ₹20,000. For instance, the Sfera Labs SPMU30X48STRATO PI SERVER UPS with a PI4B 8G is priced at ₹38,054.71.
- The method of approach revolves around the concepts of SSH and creating a automated connection between the client and the local servers such that the security barriers via the connection would not be breached.
- Use historical data, research papers, similar infrastructural study and expert judgment to estimate the time and effort required for development.
- Consider the learning curve associated with new technologies, ensuring estimates account for any training or ramp-up time.

3.1.2 Staffing

- Identify the necessary roles for the development team, such as Python developers, Testers and system administrators.
- As a specification a single employee with the adequate knowledge on python would be placed to work on the connectional aspects along with that the advisory consultant head or a project lead and at least two testers would be required to test out the operation against different various point of approach.
- Skill requirements need to be checked based on the scaling:
- High Technical Python Programmer
- Detailed Testers
- System administrators having access to the multiple servers
- Define the responsibilities and expectations for each role within the project.
- Assess the skills and experience of potential team members
- Ensure a balance of skills, experience, and expertise within the team to complement each other's strengths and weaknesses.

3.1.3 Resource Acquisition

- Identify the hardware and software requirements, such as:
- Windows 11 OS, I3 processor, and 10th generation system.
- Server Access
- Global Server Access for trial purpose
- Assess the current resources available within the organization like their local server points and determine what needs to be procured externally.
- The major constrains related to the resouces were the fact that without high level of security assurance giving access to their private server.
- Budget will lie around the range of ₹77,044.71
- Obtain necessary approvals for resource acquisition as the use of the local server would be required for the after stages.

3.1.4 Project Staff Training

- Identify the specific training needs for the development team, focusing on
- Python programming
- Multithreading
- Encryption
- SSH protocol working environment
- Linux commands and
- Paramiko library
- Research suitable training programs, workshops, or online courses
- Develop a training schedule and allocate time for team members to participate in the training programs.
- Monitor and evaluate the effectiveness of the training programs, gathering feedback from team members.

3.2 WORK PLAN

3.2.1 Work Breakdown Structure

- Break down the project into major components, such as:
- User interface design
- SSH connection implementation
- Encryption/decryption logic
- Command handling.
- Further divide each major component into smaller, manageable tasks, considering the sequence and dependencies of activities.
- Assign responsible team members or teams to each task and ensuring clarity of ownership
- Review and validate with the project internal stakeholders to ensure completeness
- Regularly update and refine the project progresses and incorporating feedback

3.2.2 Schedule Allocation

- Develop a PERT chart or project timeline that illustrates the sequence of tasks and their respective durations.
- Allocate realistic timeframes for each task, considering factors like complexity, resource availability, and potential risks.
- Include buffer time in the schedule to account for unexpected delays or issues that may arise during the project.
- Continuously monitor and update the project schedule to reflect actual progress and make necessary adjustments to meet deadlines.
- Ensure that the schedule is achievable and aligns with the overall project objectives and client expectations.
- Conduct regular progress meetings to track adherence to the schedule and identify any deviations that need to be addressed promptly.

3.2.3 Resource Allocation

- Assess the skill sets and expertise required for each task to determine the appropriate allocation of team members.
- Consider the availability and workload of each team member when assigning tasks, avoiding overburdening or underutilizing resources.
- Optimize resource allocation to maximize productivity and efficiency throughout the project.
- Focusing on the resources need for development
- Windows 11 OS, I3 processor, and 10th generation system.
- Server Access
- Global Server Access for trial purpose
- Allocate backup or contingency resources in case of unforeseen circumstances or resource unavailability.
- Establish a mechanism to reallocate resources dynamically based on shifting project priorities or urgent requirements.

3.2.4 Budget Allocation

- Develop a detailed budget that includes estimates for hardware, software, training, staffing, and other project-related expenses.
- Categorize the budget into distinct cost centers
- Some approximate prices based the requirements are:
- Travel (Managemental Needs)
- Meetings (Scheduled twice a week)
- Computing resources (Laptops and access servicing needs)
- Software tools (Operational needs)
- Special testing and simulation facilities, and (Additional Investment Salary)
- Administrative support.(Access value)
- Provide justifications for budget items, ensuring transparency and understanding of the allocation decisions.
- Monitor budget utilization regularly to track expenses against the allocated amounts and make adjustments as needed.
- Implement a cost-control mechanism to prevent budget overruns and adhere to financial constraints.

3.3 PROJECT TRACKING PLAN

3.3.1 Requirements Management

- Document and organize all project requirements in a structured and accessible manner.
- Utilize a requirements management tool to track changes, versions, and status of each requirement throughout the project lifecycle.
- Establish a process for gathering, reviewing, and validating requirements with stakeholders to ensure completeness and accuracy.
- Conduct regular requirement reviews and updates based on evolving project needs and stakeholder feedback.
- Create a traceability matrix to link requirements to specific project components and features, aiding in impact analysis and testing.

3.3.2 Schedule Control

- Monitor the project schedule regularly to track task progress and ensure adherence to the planned timeline.
- The schedule control activities by identifying the processes to be used for the purposes:
- To measure the progress of work completed at the major and minor project milestones by using basic track of working progress report
- To compare actual progress to planned progress by keeping the track of durations in the pert chart
- Communicate schedule updates and adjustments promptly to all stakeholders to manage expectations and maintain transparency.
- Implement a change management process to handle schedule modifications, seeking appropriate approvals and documenting changes.

- Conduct regular progress reviews to assess schedule performance
- Utilize project management software to automate schedule tracking and generate relevant reports for analysis and decision-making.

3.3.3 Budget Control

- Monitor actual project expenses against the budget allocation on a regular basis
- Implement expense tracking mechanisms, ensuring all expenditures are accurately recorded and categorized according to the budget structure.
- Review and approve budget-related expenses in accordance with established processes and authorization levels.
- Encourage team members to be mindful of budget constraints and optimize resource usage
- Conduct budget reviews and variance analyses, providing insights into financial trends and potential areas for cost-saving measures.

3.3.4 Quality Control

QUALITY CONTROL PROCESSES	DESCRIPTION
Quality Metrics and Criteria	Define quality metrics and criteria to evaluate performance, functionality, and security. Metrics may include response time, error rates, and adherence to security best practices.
Testing Procedures	Implement thorough testing procedures, including unit tests, integration tests, and user acceptance testing (UAT). Create test cases to validate software functionalities.
Defect Identification and Categorization	Document identified defects, categorize them based on severity (e.g., critical, major, minor), and prioritize for resolution.
Code Reviews and Inspections	Conduct regular code reviews and inspections to maintain code quality, adherence to coding standards, and best practices.
Integration of Automated Testing Tools	Integrate automated testing tools (e.g., Pytest for Python) to increase testing efficiency and accuracy. Use for automated unit and integration tests.
Continuous Monitoring and Improvement	Continuously monitor the quality control process, gather insights, and use lessons learned to improve the process iteratively.

3.3.5 Reporting

- Develop a reporting framework that outlines the types of reports, their frequency, and the intended audience for each report.
- Generate and distribute regular progress reports summarizing project status, achievements, challenges, and upcoming milestones.
- Include visual aids, charts, and graphs to enhance the clarity and impact of the reported data.
- Maintain a centralized repository for all project reports and documentation to ensure accessibility and version control.

3.3.6 Project Metrics

- Define a set of key performance indicators (KPIs) relevant to the project, such as server connections established per unit of time, successful command executions, and response times.

- Establish benchmarks or targets for each metric to measure project performance and track progress towards predefined goals.
- Utilize project management tools to automate metric tracking and generate real-time reports for timely decision-making.
- Review metrics at predetermined intervals to identify trends, patterns, and areas for improvement.

1.4 RISK MANAGEMENT PLAN

- Identify potential risks associated with the project, such as security vulnerabilities, network connectivity issues, and delays in server responses.
- Certain Potential Risks:
- High dependency on the python developer
- Tester may not be able to track all possible aspects of error
- Leak out of confidential information
- Client server crash
- Weak encryptional mechanisms
- Assess the likelihood and impact of each identified risk to prioritize them based on their significance and potential to disrupt the project.
- Develop risk mitigation strategies for high-priority risks, outlining actions to prevent or minimize their occurrence and impact.
- Certain Resolving Techniques towards Risks:
- Remove high rate dependence on the python developer
- Employee different strategic Testers to track all possible aspects of error which might lead to corruption of the private server later on
- Using Strong encryptional mechanisms
- Regularly review and update the risk management plan to incorporate new risks, assess the effectiveness of existing strategies, and make necessary adjustments.
- Communicate the risk management plan to all internal stakeholders to create awareness and alignment on risk handling strategies.
- Foster a risk-aware culture within the team, encouraging proactive risk identification, reporting, and resolution throughout the project.

3.5 PROJECT CLOSEOUT PLAN

- Define the criteria and conditions that signify successful project completion and readiness for the closeout phase.
- Conduct a thorough review of all project deliverables and outcomes to ensure they meet the specified requirements and quality standards.
- Compile comprehensive documentation, including project reports, user manuals, technical documentation, and any other relevant materials.

- Create a transition plan outlining the handover process, responsibilities, and support mechanisms for ongoing maintenance and support of the software.
- Conduct a knowledge transfer session with the maintenance and support teams, ensuring they have a clear understanding of the software's architecture, functionality, and potential issues.
- Archive project-related information and documentation in a structured and accessible format for future reference and audits.

4. TECHNICAL PROCESS PLANS

4.1 PROCESS MODEL

The process model for the project outlines the relationships among major project work activities and supporting processes.

- **Relationships among Major Project Work Activities and Supporting Processes**: The project activities will be closely interconnected. They will commence with requirement and feasibility analysis, followed by SRS design, literature review, and development activities. Frontend and backend code development activities will run concurrently and lead to integration. This integrated code will form the basis for testing, deployment, and, ultimately, report writing for project completion.
- **Flow of Information and Work Products:** Information will flow between various phases of the project, with documentation and design artifacts serving as key work products. Data about SSH server configurations will be read from the text file and will form the basis for backend development. The integrated frontend and backend code will shape the console for remote server interaction.
- **Timing of Work Products:** The primary work product in this process model will be the integrated console for interacting with remote servers. This product will be generated after the integration of frontend and backend code. It will serve as the outcome of the project, enabling users to securely log in to servers, establish connections, and execute commands such as pwd and ls.
- **Reviews to be Conducted:** Formal reviews will be conducted at major project milestones. These will include reviews for requirements, design, and acceptance testing to ensure alignment with project objectives and quality standards.
- **Major Milestones:** Key milestones will include project initiation, design completion, integration of frontend and backend code, successful testing, deployment, and project termination.
- **Baselines to be Established:** Baselines will be established at each major milestone to ensure the stability and traceability of project artifacts.
- **Project Deliverable:** The primary project deliverable will be the fully functional software with a console for interacting with remote servers, allowing secure login, connection establishment, and command execution.
- **Required Approvals:** Approvals from project stakeholders, including the client, will be necessary at various stages, particularly during project initiation and acceptance testing.

- **Project Initiation and Termination Activities**: Project initiation will involve requirement and feasibility analysis, setting the foundation for the entire project. Project termination, marked by report writing, will involve documenting the project's achievements, outcomes, and lessons learned.
- **Tailoring of Process Model**: Our organisation's standard process model will be tailored to align with the Agile development methodology chosen for this project, allowing for flexibility and iterative development.

4.2 METHODS, TOOLS AND TECHNIQUES

- **Development Methodology:** Agile methodology would be adopted for flexibility and iterative approach.
- **Programming Language:** Python would chosen for its compatibility with SSH (Paramiko library) and rapid development capabilities.
- **Design:** Object-oriented principles will guide the design for modularity and maintainability.
- **Testing:** Comprehensive testing including unit, integration, and user acceptance testing would be carried out.
- **Documentation**: Clear and concise documentation would be maintained throughout the project.
- **Security:** RSA encryption will be implemented for secure data transmission.

4.3 INFRASTRUCTURE

- **Hardware:** Windows 11 systems with Intel Core i3 processors (10th generation or higher) to accommodate development needs.
- **Operating System:** Windows 11 for compatibility with development tools.
- **Network:** Reliable internet connectivity for collaboration and remote server access.
- **Software Tools:** Visual Studio Code for Python development, alongside necessary libraries like Paramiko.

4.4 PRODUCT ACCEPTANCE

- **Customer Acceptance Plan:** Customer acceptance will be a structured process. A formal plan will be prepared, specifying the steps, criteria, and responsibilities for acceptance.
- **Objective Criteria:** The acceptability of deliverables will include functionality, performance, security, and usability checking. For example, functionality will be assessed by verifying that all SSH commands execute correctly and securely.
- **Formal Agreement:** A formal agreement outlining the acceptance criteria will be established. This agreement will be signed by representatives of our IT organization and the customer, ensuring mutual understanding and commitment to the criteria.
- **Technical Processes and Tools**: The following processes will be included for deliverable acceptance. These include:

 Testing: Rigorous testing, including unit testing, integration testing, and user acceptance testing, will be conducted to validate the functionality and security of the console.

- **Demonstration:** A live demonstration will be provided to the customer, showcasing how the console securely connects to remote servers and executes commands.
- **Analysis:** In-depth analysis of the codebase and security measures will be performed to ensure compliance with industry standards.
- **Inspection:** Code inspection and review will be conducted to identify and rectify any potential issues.

5. SUPPORTING PROCESS PLANS

5.1 CONFIGURATION MANAGEMENT

In a project involving server management, effective SSH configuration management is vital for security and consistency.

- *Configuration Identification:* Central Git repository with unique identifiers.
- *Configuration Control:* Access controls, change management, and backups.
- *Status Accounting:* Records and audit trails for tracking changes.
- *Evaluation:* Periodic security audits and risk assessments.
- *Release Management:* Versioning, release notes, and automation.
- *Initial Baselining:* Documenting and storing initial configurations.
- *Change Request Logging:* Establish request system, log changes, analyze impact.
- *Change Control Board:* Review and approve changes.
- *Tracking Changes:* Monitor progress, assign responsibilities, maintain a change log.
- *Notification Procedures:* Communicate baseline changes, updates, and impacts.

These practices ensure secure, reliable, and consistent SSH connections while promoting accountability and transparency in infrastructure management.

5.2 VERIFICATION AND VALIDATION

Verification and validation encompass all aspects of software development, ensuring adherence to requirements and standards.

- *Tools and Techniques*: Use automated testing tools, manual reviews, simulations, and modeling for comprehensive assessment.
- *Responsibilities*: Development teams ensure traceability, quality assurance teams perform reviews, and dedicated testing teams handle testing activities.
- *Organizational Relationships*: Verification and validation activities remain independent but closely aligned for iterative improvement, conducted by impartial review teams.
- Verification Techniques: Employ traceability for requirements to implementation linkage, milestone and progress reviews for compliance, peer reviews for code quality, and prototyping for design validation.
- Validation Techniques: Conduct comprehensive testing, demonstrate software functionality to stakeholders, analyze system behavior and outputs, and ensure compliance with predefined

standards.These strategies ensure a rigorous assessment and validation process, enhancing software quality and compliance with requirements and standards.

5.3 DOCUMENTATION

Plans for Generating Non-Deliverable and Deliverable Project Documentation:

- Project Documentation Management Overview:
  - Non-Deliverable Documentation: Internal project documents for project team reference, not included in final deliverables.
  - Deliverable Documentation: Shared with external stakeholders, essential for understanding and testing the software.
- Responsibilities:
  - Developers: Provide technical details and create technical specifications, code documentation.
  - Users: Contribute user requirements and create user-focused documents like guides and manuals.
  - Testers: Define testing requirements, develop test cases, and create testing documentation including test plans.
  - Project Management Team: Oversee the documentation process, coordinate generation, review, and distribution.
- List of Documents to Prepare:
  - Non-Deliverable Documentation: Project Plan, Project Schedule, Change Control Procedures, Internal Meeting Minutes.
  - Deliverable Documentation: User Guide, Installation Guide, Testing Plan, Test Cases, Code Documentation, Technical Specifications.
- Controlling Templates/Standards:
  Specify templates/standards for each document (e.g., company specific templates for user guides, coding standards for code documentation).

- Document Preparation & Review:
  - Developers create code documentation; users generate user guides.
  - Define individuals/teams responsible for reviewing each document.
- Due Dates: Set deadlines for review copies and initial baseline versions.
- Distribution List & Quantities:
  - Determine who needs access to review copies and baseline versions of each document.
  - Specify quantities required for distribution.

  By establishing these plans and responsibilities, the project documentation process will be well organized, efficient, and involve all relevant stakeholders in its creation and review, ensuring clarity and quality in project documentation.

5.4 QUALITY ASSURANCE

- *Quality Objectives:*

1. Reliability and Consistency: Preventing downtime and data loss.

2. Performance Optimization: Resource optimization for responsive user experiences.

3. Usability Enhancement: Intuitive, user-friendly interface.

4. Maintainability: Modular codebase for easy updates.

5. Security Compliance: Stringent measures to protect against threats.

- *Quality Assurance Procedures:*

1. Testing: Comprehensive testing (functional, performance, security, usability).

2. Usability Evaluation: Assessing user experience.

3. Code Review: Peer reviews for code quality.

4. Security Audits: Regular audits, including penetration testing.

- *Relationships*:

- QA: Ensures adherence to standards and objectives.

- Verification and Validation (V&V): Validates compliance with requirements.

- Review and Audit: Part of QA, covers code, security, and performance.

- Configuration Management: Maintains code integrity and tracks changes.

- System Engineering: Aligns with QA to meet system requirements.

- Assessment Processes: Validate compliance with quality objectives and standards.

5.5 REVIEWS AND AUDITS

Schedule:

Review/Audit Type	Schedule and Frequency	Resources	Processes and Procedures
Customer Project Reviews	Periodic throughout the project	Project Team, Customers	Scheduled meetings with customers to review project progress and align expectations.
Management Progress Reviews	Regular intervals (for e.g., monthly)	Project Manager, Team	Project Manager leads reviews, assesses progress, and identifies potential issues.
Developer Peer Reviews	Throughout the	Development Team	Developers review each other's code for quality, adherence to coding standards, and bug identification.

	development phases		
Quality Assurance Audits	As needed, based on QA plan	QA Team	QA Team conducts audits to ensure adherence to quality procedures and standards.
Customer Conducted Reviews/Audits	At customer discretion	Customers, Project Team	Customers may conduct their reviews or audits based on their specific requirements.
Regulatory Agency Reviews	As required by regulatory agencies	Regulatory Agencies	Compliance with external regulations is verified through reviews conducted by relevant regulatory agencies.

5.6 PROBLEM RESOLUTION

Teams:

- *Problem Reporting Team:* Identifies and reports issues.
- *Problem Analysis Team:* Analyzes problems to determine their nature and severity.
- *Problem Resolution Team:* Resolves identified issues.

Resources:

- *Problem Tracking Tool:* Dedicated software for problem logging and management.
- *Communication Channels:* Facilitating efficient issue reporting and resolution.
- *Documentation:* Captures detailed problem reports, analysis findings, and resolution procedures.

Methods, Tools, Techniques, and Procedures:

- *Problem Reporting:* Any team member can report issues using the Problem Tracking Tool by completing a problem report form with issue details.
- *Problem Analysis:* They will investigate
- reported issues using debugging tools, logs, and documentation. They analyze the issue, categorize severity, and prioritize.
- *Problem Resolution:* They will address and fix reported issues using development environments and version control systems. They will develop fixes, test them, and integrate them into the project.

Roles:

- *Development*: Implements fixes.
- *Configuration Management:* Ensures version control and proper integration.
- *Change Control Board*: Reviews and approves changes related to problem resolution.
- *Verification and Validation*: Ensure problem resolution doesn't introduce new issues.

Tracking Effort: Separate tracking of effort for problem reporting, analysis, and resolution enables identifying rework areas and informs process improvement initiatives.

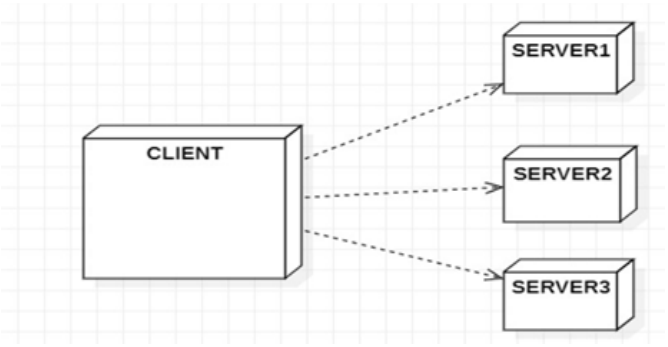
5.7 SUBCONTRACTOR MANAGEMENT

- ***Subcontractor Selection Criteria***: Subcontractors chosen based on technical expertise, proven track record, sufficient resources, financial stability, compliance with regulations, and communication skills.
- ***Subcontractor Management Plan***: For each subcontract, a dedicated plan will be crafted covering requirements, technical progress monitoring, budget and schedule control, product acceptance criteria, risk management, and specific subcontractor responsibilities.
- ***Comprehensive Approach***: This plan ensures effective management and oversight of subcontractors by defining clear expectations, monitoring progress, and addressing risks. It promotes successful collaboration and the achievement of project objectives across all subcontractor relationships.

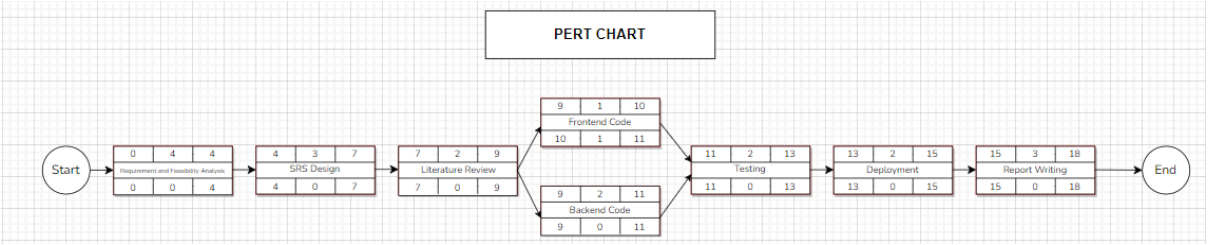
5.8 PROCESS IMPROVEMENT

- Periodic project assessments will be conducted using Key Process Indicators.
- Cross-functional Assessment and Process Improvement teams will be formed.
- Improvement areas including bottlenecks and communication issues will be identified.
- Process Improvement and Problem Resolution Plans will be assigned.
- Problem resolution insights for process enhancement will be used.
- Non-disruptive improvements and align with organization-wide initiatives will be prioritized.
- Collaboration with the organization for broader project process enhancements will be considered in future.

ANNEXURE A



Diagrammatic Representation of the Project Architecture



PERT Chart

EXPERIMENT 8

AIM: To prepare the workflow of the project

THEORY:

A workflow diagram typically comprises several key elements that collectively represent the sequential steps and dependencies within a process. When describing these elements in the third person, one might articulate as follows:

1. Start and End Points: The diagram begins with a clear starting point and concludes with a distinct endpoint, indicating where the process initiates and terminates.

2. Process Steps: The process involves multiple steps or tasks, each represented by specific shapes or symbols. These steps are sequenced to illustrate the flow of activities.

3. Flow Arrows: Arrows are used to indicate the direction of flow between different steps, offering a visual representation of the sequence in which tasks are performed.

4. Decision Points: Decision points are integrated into the workflow, typically depicted as diamond shaped symbols. These points signify moments in the process where a decision must be made, influencing the subsequent path.

5. Connectors: Connectors link different parts of the workflow, indicating relationships or dependencies between various steps and ensuring a cohesive representation of the entire process.

6. Roles and Responsibilities: Information about the individuals or roles responsible for each task is included, offering clarity on who performs specific activities within the workflow.

7. Data or Information Flow: Arrows or lines with labels demonstrate the flow of data or information between different steps, showcasing how information moves throughout the process.

8. System Inputs and Outputs: If the workflow involves interactions with systems or external entities, inputs and outputs are identified to highlight the exchange of information between the process and external elements.

9. Timing and Milestones: The diagram may include elements indicating timing, milestones, or critical points in the process, providing a timeline perspective on the overall workflow.

10. Annotations and Notes: Annotations or notes may be added to provide additional information or context, offering insights into specific aspects of the process.

RESULT:

A[Start] --> B[Read text file with server info]

B --> C[Determine number of servers]

C --> D[Initialize multithreading]

D --> E[Connect to servers using Paramiko]

E --> F[User inputs commands]

F --> G[Send commands to servers]

G --> H[Servers process commands]

H --> I[Return encrypted output]

I --> J[Decrypt output]

J --> K[Display output to user]

K --> L[End]

EXPERIMENT 9

AIM: To make SLIP chart, Timeline chart and Ball chart for the project

THEORY:

SLIP Chart: A slip chart, in the context of software project management, is a graphical representation that shows the schedule slippage or delays in completing various tasks or activities in a project. It is a tool used to track the progress of project activities over time and compare planned versus actual completion dates. Here's how a slip chart is typically constructed:

X-Axis (Horizontal Axis): Represents time, often in days or weeks.

Y-Axis (Vertical Axis): Represents activities or tasks in the project.

Uses of Slip Chart in Software Project Management:

- Schedule Monitoring:

  Planned vs. Actual Comparison: Slip charts provide a visual representation of how the actual progress compares to the planned schedule. It helps project managers quickly identify areas where delays are occurring.

- Decision Making:

  Early Warning System: By monitoring slip charts regularly, project managers can identify potential issues early on and make informed decisions to mitigate risks and prevent further delays.

- Resource Allocation:

  Resource Management: Slip charts can help in evaluating whether resources are adequately allocated to tasks. If certain activities consistently experience delays, it may indicate a need for additional resources or adjustments in resource allocation.

Timeline Chart: A timeline chart in software project management is a visual representation of project activities and their durations over time. Also known as a Gantt chart, it provides a comprehensive view of the project schedule, showing when each task or activity is planned to start and finish. The timeline chart is a widely used tool for project planning, monitoring, and communication. Key features of a timeline chart include:

• Task Bars: Each task or activity is represented by a horizontal bar on the chart. The length of the bar corresponds to the duration of the task, and the position on the timeline indicates when the task is scheduled to start and finish.

• Time Axis: The chart has a horizontal axis representing time, typically broken down into days, weeks, or months. This axis allows project managers and team members to see the project's timeline and schedule at a glance.

• Dependencies: Timeline charts often use arrows or other symbols to represent task dependencies. This visual representation helps project managers understand the sequence of activities and the relationships between them.

• Milestones: Important project milestones, such as key deliverables or deadlines, are often marked on the timeline. Milestones provide clear points of reference for project progress and completion.

• Resource Allocation: Some timeline charts include information about resource allocation, indicating which team members are responsible for specific tasks during specific timeframes.

Ball Chart: A somewhat more striking way of showing whether or not targets have been met is to use a ball chart. In this version of the ball chart, the circles indicate start and completion points for activities. The circles initially contain the original scheduled dales.

RESULTS:

1. SLIP Chart:

S. No.	Milestones
1.	Define Project scope and objectives
2.	Develop user interface design
3.	Implement backend functionality

• On a weekly or monthly basis milestones are plotted on a grid to show when they are scheduled to occur. In the example 0 represents the start of the project. In the baseline schedule milestone M1 is due to occur in week 2; M2 in week 5 and M3 in week 7.

• At the end of the first week of work, all three milestones are on schedule, by the end of week two they are all running a week late. At the end of week three M1 finishes a week late and the other milestones are still running one week late.

• In week four M2 is back on schedule but M3 is still running a week late. M2 finishes on time in week five but by week six M3 has been delayed a further week.

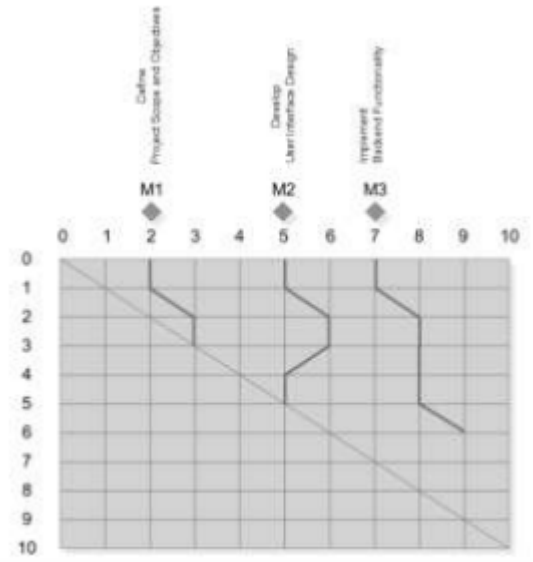


Figure: SLIP Chart

2. Timeline Chart: Creating a timeline chart for the development of a social media app involves breaking down the process into key phases and milestones:

1. Conceptualization and Planning:

• Duration: 1-2 months

• Define the app's purpose, target audience, and unique features, conduct market research and competitor analysis, create a basic wireframe and outline core functionalities.

2. Design Phase:

• Duration: 2-3 months

• Develop detailed app wireframes and user flows, create the user interface (UI) and user experience (UX) design, obtain feedback from potential users and iterate designs accordingly.

3. Development Kickoff:

• Duration: 1 month

• Set up the development environment and infrastructure, define the technology stack (backend, frontend, database), develop a Minimum Viable Product (MVP) with basic features.

4. Backend Development:

• Duration: 3-6 months

• Create the server-side architecture, implement user authentication and authorization, develop the database structure, build APIs to support frontend functionality.

5. Frontend Development:

• Duration: 3-6 months

• Develop the user interface based on approved designs, implement client-side logic and user interactions, integrate with backend APIs.

6. Testing:

• Duration: 2-3 months

• Conduct unit testing, integration testing, and system testing, identify and fix bugs, perform user acceptance testing (UAT) with a select group of users.

7. Beta Release:

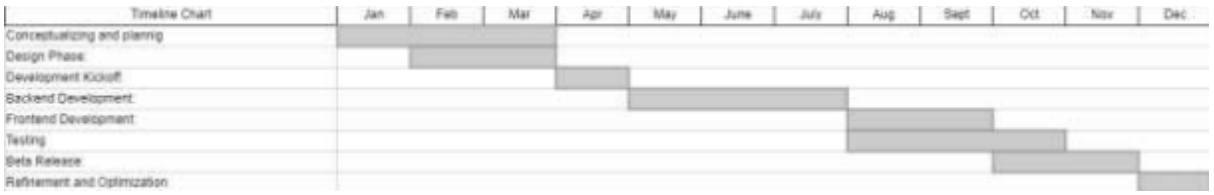
• Duration: 1-2 months

• Release a beta version to a limited audience for testing and feedback, gather insights on performance, usability, and any issues.

8. Refinement and Optimization:

• Duration: Ongoing

• Analyze user feedback and make necessary improvements, optimize performance and fix any remaining bugs.



3. Ball chart: