The Bombay Salesian Society's

DON BOSCO INSTITUTE OF TECHNOLOGY

(An Autonomous Institute Affiliated to University of Mumbai)

A PROJECT REPORT ON

"DEPLOYX"

SUBMITTED BY:

CHETAN CHAUDHARI 10
NISCHAY CHAVAN 14
PARTH SHIKHARE 53

UNDER THE GUIDENCES OF:

Dr. Vaishali Kavathekar

DEPARTMENT OF INFORMATION TECHNOLOGY

(2025-2026)

CERTIFICATE

This is to certify that the project entitled **"DEPLOYX"** is a bona fide work of **CHETAN CHAUDHARI, NISCHAY CHAVAN and PARTH SHIKHARE** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of "**TEIT"** in **"Mini Project – 2 A Web Based Business Model"**.

(GUIDE SIGNATURE)

Dissertation Approval Certificate

This project report entitled Website on DeployX by Chetan Chaudhari, Nischay Chavan and Parth Shikhare, is approved for the degree of Bachelor of Engineering in Information Technology.

Examiners

Name:-

Date:-

Place:-

Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited or from whom proper permission has not been taken when needed.

Chetan Chaudhari – 10
Nischay Chavan – 14
Parth Shikhare -53

Date:

Abstract

DeployX is an advanced bulk software deployment system designed to streamline the distribution, monitoring, and management of applications across multiple remote devices. The system integrates a visually engaging, cyberpunk-inspired frontend built with React, Vite, and Tailwind CSS with a robust backend powered by FastAPI, enabling real-time control over connected systems. DeployX provides an interactive dashboard for system administrators to perform tasks such as remote shell execution, file system management, process and service control, and detailed system information monitoring. The platform also supports QR code-based access to project demos and media, ensuring seamless onboarding and visualization of deployment workflows.

The system emphasizes automation, efficiency, and security, making it suitable for enterprise environments requiring centralized software distribution. By combining real-time interactivity, role-based dashboards, and scalable deployment mechanisms, DeployX ensures rapid, reliable, and controlled software delivery to multiple endpoints. Additionally, it facilitates proactive system monitoring, quick fault detection, and resource allocation, reducing manual intervention and minimizing downtime. The futuristic UI/UX design enhances user engagement while simplifying complex deployment tasks, bridging the gap between advanced system administration and intuitive usability. DeployX represents a convergence of automation, real-time monitoring, and visually immersive design, providing a comprehensive solution for modern IT infrastructure management.

Keywords: Bulk Software Deployment, Remote System Management, React, FastAPI, Real-Time Monitoring, Automation, Dashboard, QR Code Integration, Cyberpunk UI/UX

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Managing multiple computers in an organisation, lab environment, or distributed network is difficult and time-consuming when administrators need to perform tasks manually on each system. Activities such as installing software, running system commands, monitoring system health, transferring files, handling processes, and troubleshooting require physical access or individual remote sessions for every machine.

Traditional remote management tools are expensive, platform-dependent, or lack real-time control. Existing solutions do not provide a unified interface to manage a fleet of devices with instant output, automation capabilities, and secure communication. Therefore, a centralised remote control system is needed that can manage multiple computers efficiently and securely from a single dashboard.

1.2 Scope of the project

The focus of DeployX is to build a web-based remote management and automation platform capable of controlling multiple systems in real-time using lightweight agents. The system will support the following:

- Real-time terminal command execution across remote devices
- Live monitoring of CPU, RAM, Disk, and Network
- File system management (upload, download, rename, delete, directory navigation)
- Process and service management
- Secure agent authentication and logging
- Cross-platform support (Windows, Linux, macOS)
- Communication over LAN or the Internet
- Persistent connectivity with auto-start and heartbeat signals

The scope includes creating an interactive dashboard using React.js, a scalable backend using Node.js, FAST API, and PostgreSQL, and a cross-platform Python agent.

1.3 Current Scenario

In the current environment, system administrators rely on manual methods or third-party remote access tools such as AnyDesk, TeamViewer, or SSH/RDP to manage each computer individually. These approaches require repeated logins, consume time, and are inefficient when handling a large number of systems at once. Existing remote tools focus mainly on screen sharing, troubleshooting, or remote access, but they do not provide:

- Batch command execution
- Centralized file management
- Real-time tracking of system performance
- Remote process/service automation
- Multi-device orchestration from a single interface

As a result, managing large networks becomes complex and lacks automation and scalability.

1.4 Need for the proposed System

The proposed DeployX system is essential because it overcomes the limitations of the existing solutions by offering:

- Centralized control of multiple remote machines from one dashboard
- Automation of repetitive administrative tasks
- Real-time command execution with instantly streamed output
- Better performance insights through live monitoring statistics
- Secure communication with authentication and activity logging
- Reduced operational time and human effort
- Scalability and cross-platform compatibility

DeployX improves efficiency, enhances control, and provides a modern platform for real-time management in IT labs, corporate networks, data centers, and distributed computing environments.

Chapter 2

Review of Literature

2.1 Summary of the investigation in the published papers

Investigation of the existing literature, online resources, and real-world remote-management applications shows that remote system control and monitoring are an important requirement in modern distributed computing environments. Research papers on fleet-management, distributed system monitoring, and remote job execution highlight the need for centralized platforms capable of managing multiple connected devices. These works commonly utilize agent-based architectures in which lightweight clients communicate with a central server to exchange status information and execute commands. They also emphasize issues such as secure communication, real-time data transmission, scalability, and fault-tolerant design.

Existing commercial remote-access tools such as TeamViewer, AnyDesk, and Remote Desktop Protocol (RDP) provide remote access and screen-sharing features, file transfer, and cross-platform support. However, they mainly focus on individual machine access rather than automated control of multiple systems simultaneously. They lack capabilities like batch command execution, real-time command output streaming, remote process and service management, system resource monitoring dashboards, and automated orchestration across a fleet of devices. Additionally, many academic fleet-management solutions are limited

to IoT devices, vehicles, or robotics systems rather than general computing environments, and typically do not provide OS-level file or process control.

From the overall investigation, it is evident that a gap exists between remote desktop applications and automation-based fleet-management systems. There is no widely adopted unified solution that provides secure, real-time monitoring and execution, full system-level control, and centralized orchestration for multiple remote computers across different operating systems. This supports the need for a system like DeployX, which integrates remote command execution, file management, performance monitoring, process control, and secure agent-based communication within a single web-based interface. The findings therefore justify the development of DeployX as a more advanced, scalable, and automation-focused remote management solution compared to currently available tools and approaches.

Chapter 3

Analysis and Design

3.1 Methodology / Procedure adopted

The development of DeployX follows a structured methodology aimed at designing a scalable, secure, and real-time remote management system using a MERN-stack architecture supported by cross-platform Python agents. The system is analyzed and designed to provide seamless communication between the web-based control panel and remote machines, enabling automated operations, monitoring, and command execution.

3.1.1 Requirement Analysis

A detailed study of existing tools and research findings was conducted to identify the limitations of current remote-access solutions and the need for improved automation and multi-device orchestration. The system requirements were divided into functional and non-functional categories. Functional requirements include remote terminal command execution, real-time system performance monitoring, file upload/download operations, process and service management, and secure authentication. Non-functional requirements include security, cross-platform compatibility, reliability, scalability, and real-time communication performance.

3.1.2 Data Flow and Process Analysis

When an administrator triggers an action from the dashboard, the request is sent to the backend server, which authenticates and forwards the command to the selected agent(s). The agent executes the command locally, collects results, and streams the output back to the server and then to the frontend. For monitoring, each agent periodically sends performance metrics (CPU, RAM, disk, network) using heartbeat signals. All interactions are logged to maintain system transparency and auditability.

3.1.3 Technology Analysis

The MERN stack was chosen for its performance, scalability, and ability to handle real-time interactions. React provides dynamic UI rendering, Node.js enables high concurrency, Express simplifies API development, and PostgrSQL supports flexible document-based data storage. Python was selected for the agent due to its cross-platform capability, extensive OS-control libraries, and ease of integration with system processes.

3.1.4 Security Measures

Security is enforced through token-based authentication, encrypted communication between the backend and agents, access permissions, and action logging. Unauthorized commands are prevented through defined execution policies and agent identity verification.

3.1.5 Performance and Scalability Considerations

The system is designed to scale horizontally by allowing multiple agents to connect concurrently and enabling distributed execution. The use of WebSockets and event-based architecture ensures efficient communication without heavy overhead. Real-time streaming and asynchronous processing improve responsiveness under high load conditions.

3.2 System architecture /Design

DeployX follows a client-server model consisting of:

- Frontend (React.js): Provides an interactive dashboard for administrators to monitor status, view logs, and issue commands.

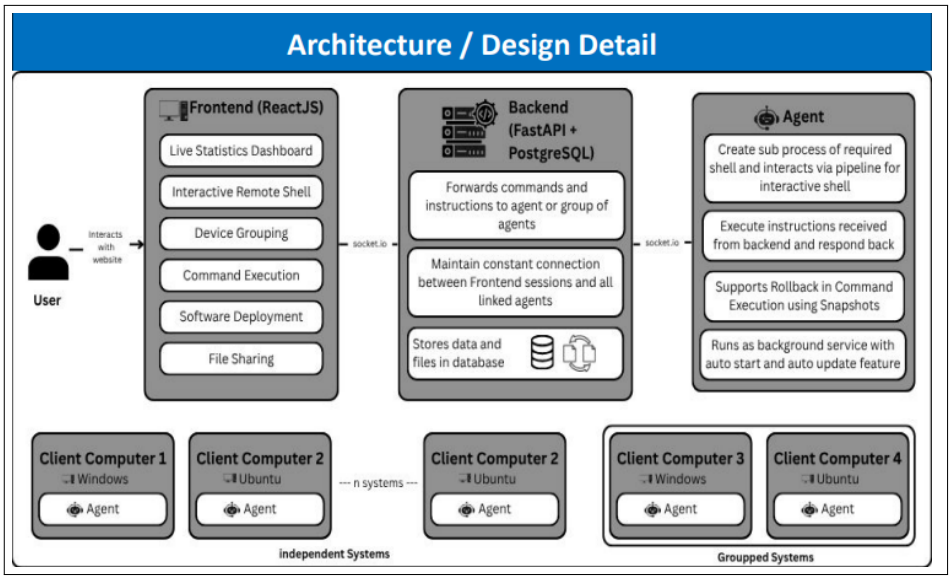

Figure 3.1: System architecture / design

- Backend (Node.js + FAST API + PostgreSQL): Handles user authentication, manages agent connections, stores logs, processes commands, and coordinates execution.
- Agent (Python script): Runs on each managed machine, maintains a persistent connection with the backend server, executes received tasks, collects system metrics, and returns results in real time.

The system utilizes WebSockets for bidirectional real-time communication, enabling immediate command execution and instant streaming of output (stdout/stderr). MongoDB stores command logs, machine identifiers, and system information to ensure traceability and security. The architecture ensures modularity, allowing new features to be added without disrupting core functionality.

Chapter 4

Implementation

4.1 Introductory UI

Figure 4.1: Landing Page

For returning users, the greets them with a ”Welcome Back!” message, emphasizing the platform’s purpose of ensuring systems remain operational anytime and anywhere. The sign-in panel includes fields for entering a username and password, alongside a clearly visible ”Login” button for secure access. Additionally, users can log in via social platforms like Google, providing more convenience. The overall interface is modern, clean, and responsive, designed

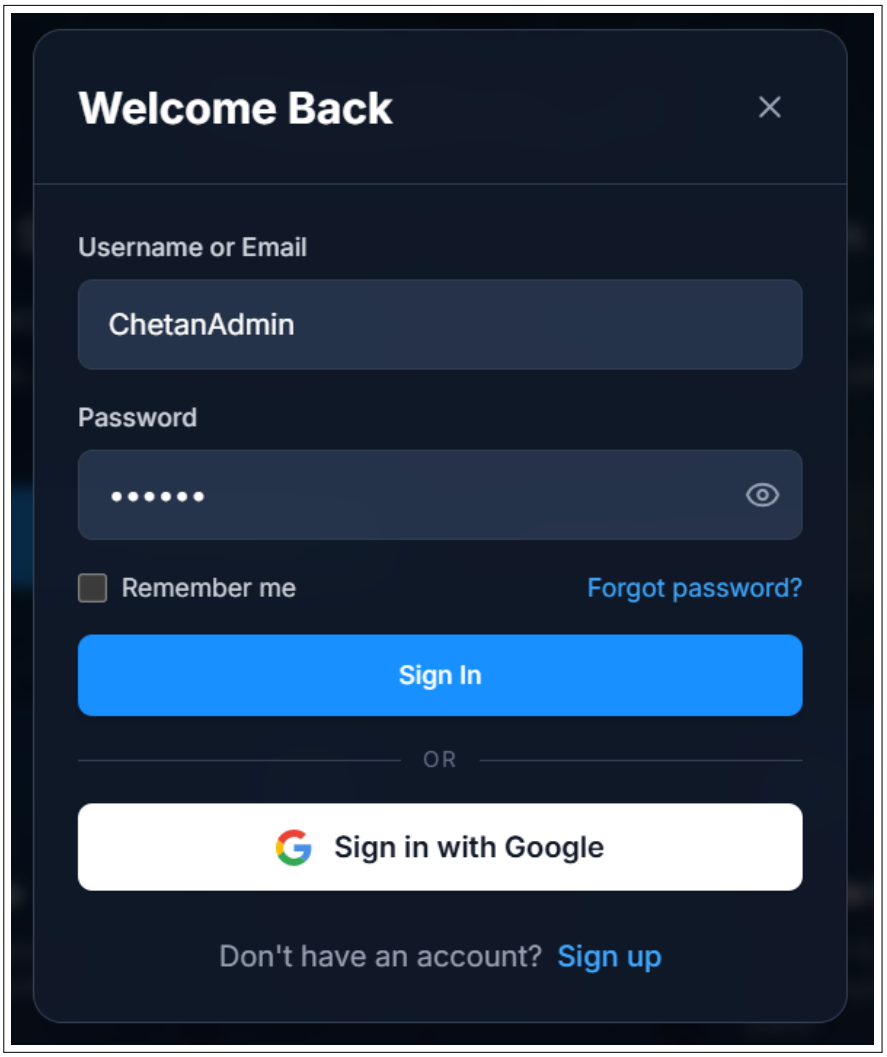

Figure 4.2: Login Page

for user-friendliness while maintaining clarity in functionality.

4.2 Dashboard

The dashboard interface of DeployX provides a clear and real-time overview of the entire remote management environment. It displays key statistics such as the total number of connected devices, overall system health score, command execution status, and deployment success rate, allowing administrators to assess the performance and availability of systems instantly. Visual indicators like graphs, percentages, and status badges help quickly identify issues such as offline devices, failed tasks, and performance degradation across the network. The interface also showcases live status tracking of activities like device connections, command completions, and deployment results, making remote

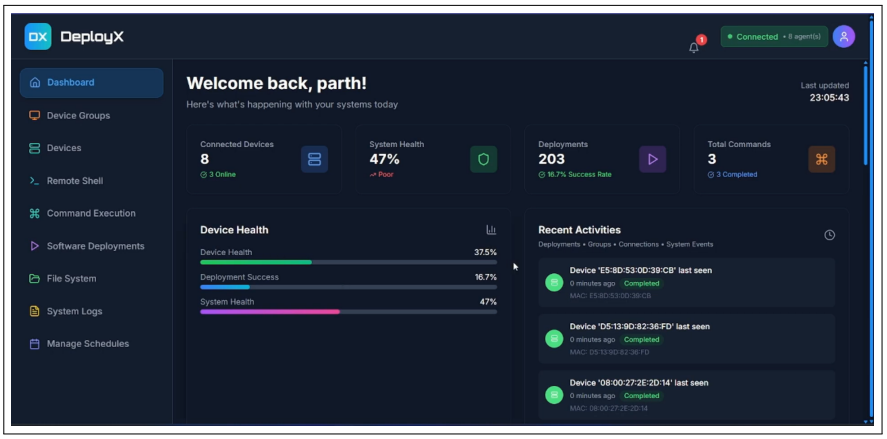

Figure 4.3: Dashboard 1

operations fully transparent and easy to monitor.

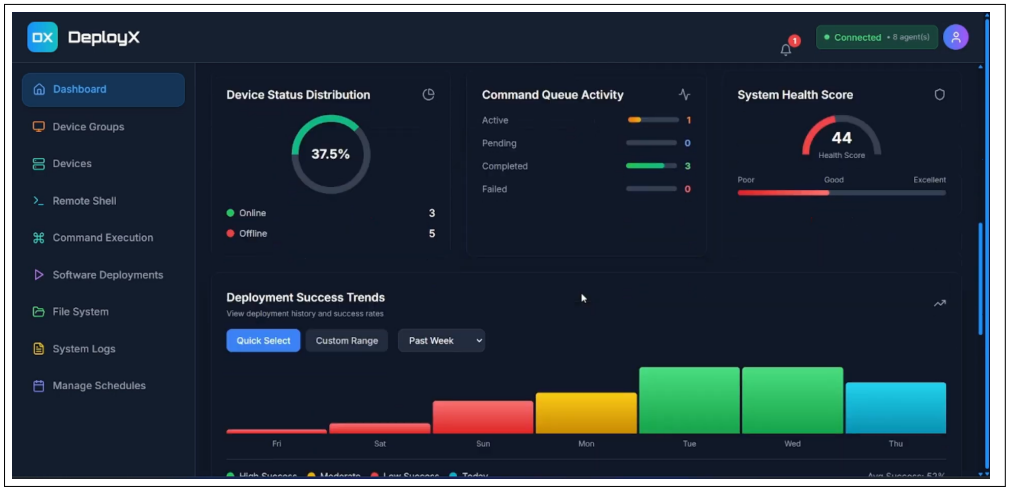

Figure 4.4: Dashboard 2

Additionally, the dashboard includes analytical components such as device status distribution, command queue activity, and deployment success trends across different timeframes. These insights help in decision-making, resource allocation, and understanding long-term behavior of deployed systems. Designed using React.js and Tailwind CSS, the UI ensures smooth navigation through major functionalities including device groups, remote shell access, command execution, software deployment, file system management, system logs, and scheduling. Overall, the dashboard acts as a central control hub that enhances usability, improves operational efficiency, and supports fast response to any system-level issues.

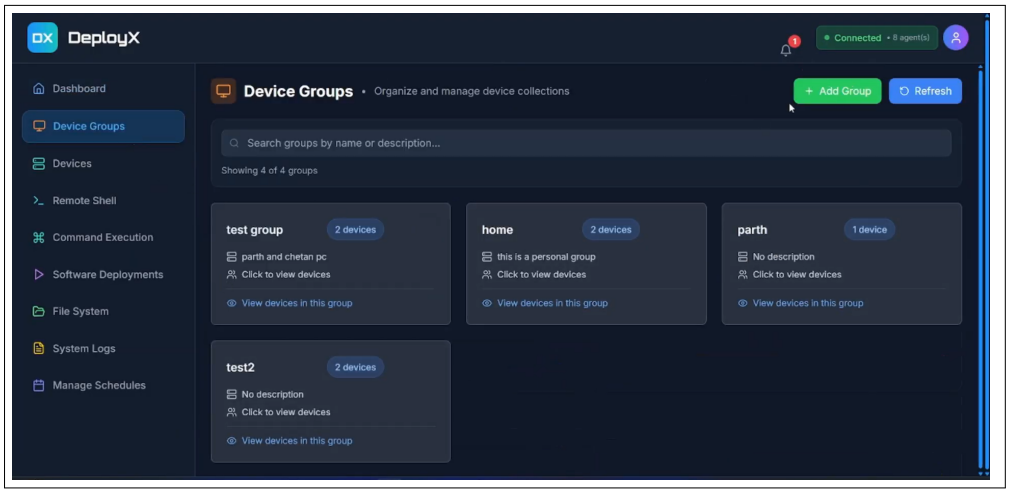

Figure 4.5: Device Groups

4.3 Device Groups

The displayed interface represents the Device Groups section of the DeployX system, where users can organize multiple remote machines into logical collections for simplified management. This screen allows administrators to create, search, and manage device groups based on usage, location, team, or project requirements. Each group card shows the group name, number of devices assigned, a short description, and a link to view the devices within that group. The ability to organize devices into groups enables efficient batch operations, such as executing commands or deploying software to multiple systems simultaneously.

The interface includes options like **Add Group** and **Refresh** at the top right, providing quick access to group creation and real-time updating of system data. Search functionality supports filtering groups by name or description, improving scalability when handling a large set of remote devices. With a clean and user-friendly layout, the device grouping feature enhances operational productivity by making remote system management more structured, flexible, and intuitive.

4.4 Devices

The Devices Management screen provides a centralized interface for monitoring and controlling all remote machines connected to DeployX. It displays a list of devices with details such as hostname, IP address, operating system, assigned device groups, and last active timestamp. A quick status summary at the top in-

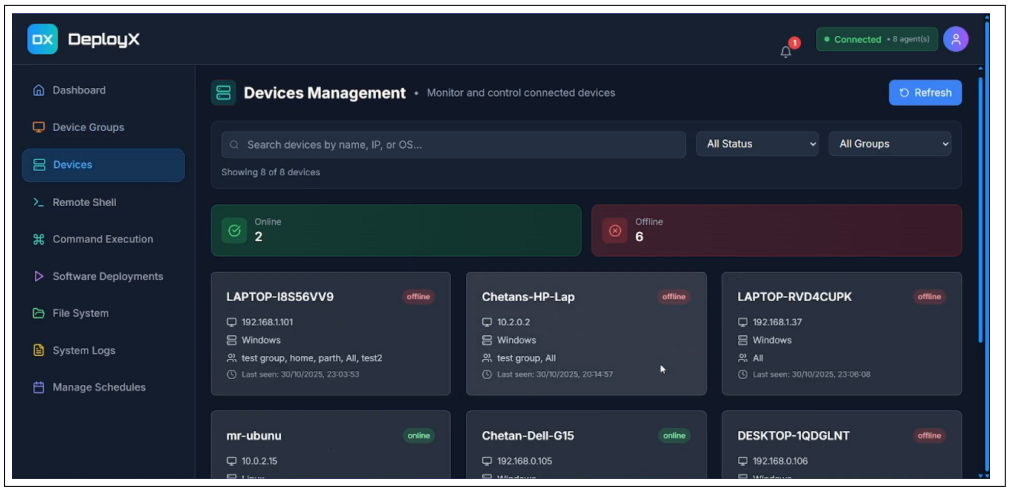

Figure 4.6: Devices

dicates how many devices are currently online or offline, helping administrators assess availability at a glance. Each device card includes real-time status indicators that allow users to visually identify system connectivity and take action accordingly. The search and filter options enable easy retrieval of devices based on name, IP, operating system, status, or group, making management efficient even in large-scale environments.

This screen enables granular control over remote systems by allowing administrators to select individual devices for further operations, such as command execution, remote shell access, file system management, or deployment tasks. The presence of refresh functionality ensures that device data is always up to date and reflects real connectivity conditions. With a structured layout and modern design, the Devices Management interface serves as a critical core component of DeployX, supporting fast decision-making and smooth remote administrative workflows.

4.5 Remote Shell

The displayed interface represents the Interactive Remote Shell feature of DeployX, which allows administrators to execute system-level commands directly on remote machines in real time. The interface provides a dropdown selection to choose the target device (agent) and supports switching between command environments such as CMD, Bash, or PowerShell depending on the operating system. Once connected, the shell opens a real-time terminal session where commands can be executed as if the user were physically present at that machine. The command output streams live on the screen, enabling immediate

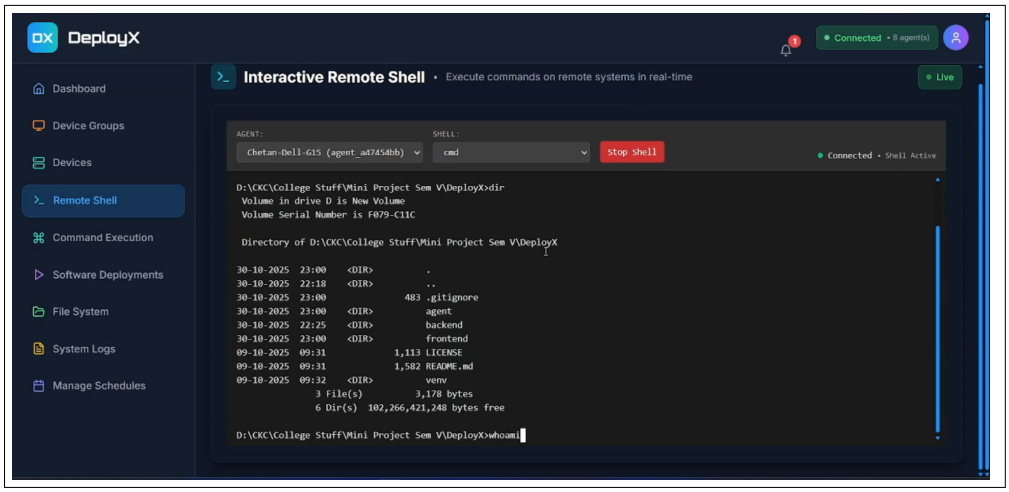

Figure 4.7: Remote Shell

feedback and verification.

The interface includes controls such as **Stop Shell** to terminate the session instantly for safety and security. Live connection indicators display the shell status and connection activity, ensuring operational transparency at all times. This feature is highly useful for remote troubleshooting, software configuration, automation, and maintenance tasks without requiring remote desktop access or manual physical interaction. The Remote Shell enhances productivity by enabling secure and controlled CLI-based access across multiple distributed systems on the network.

4.6 Command Execution

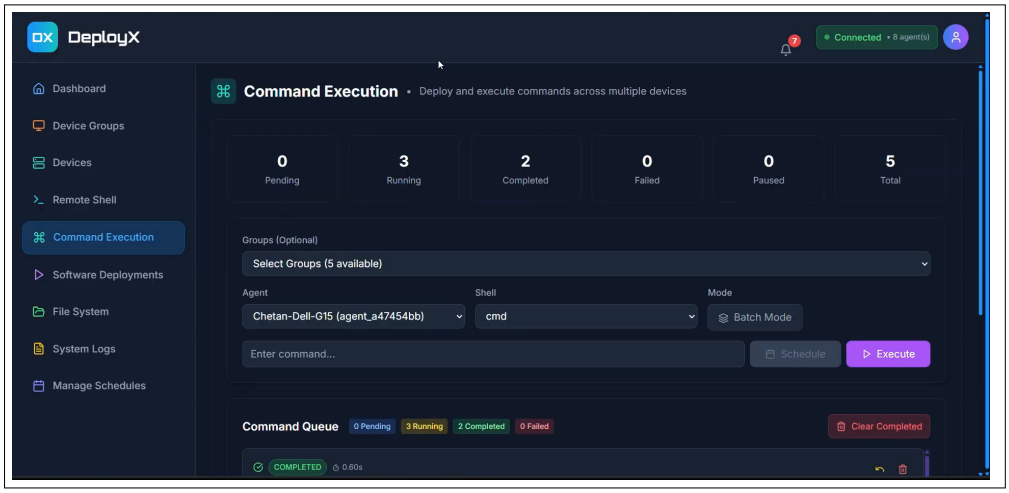

Figure 4.8: Command Execution Interface

The Command Execution interface in DeployX enables administrators to deploy

and execute system commands on remote machines or groups of devices in a centralized and efficient manner. At the top of the screen, the system provides a real-time summary of execution statuses, including the number of pending, running, completed, failed, and paused commands. This helps users monitor the progress and success rate of various operations across multiple systems simultaneously. The interface supports selecting individual devices or predefined device groups, allowing batch execution without needing to perform actions manually on each machine.

The command input section allows users to choose the appropriate shell environment (CMD, Bash, or PowerShell) based on the target system's operating system. Users can switch between single execution mode and batch mode depending on whether the command needs to be deployed to one or many devices. Once executed, each command is added to the Command Queue, which visually displays real-time status updates along with execution logs. The interface includes controls such as **Execute**, **Schedule**, and **Clear Completed** to manage command lifecycles effectively. This feature significantly enhances productivity, automation, and remote management capabilities within distributed environments.

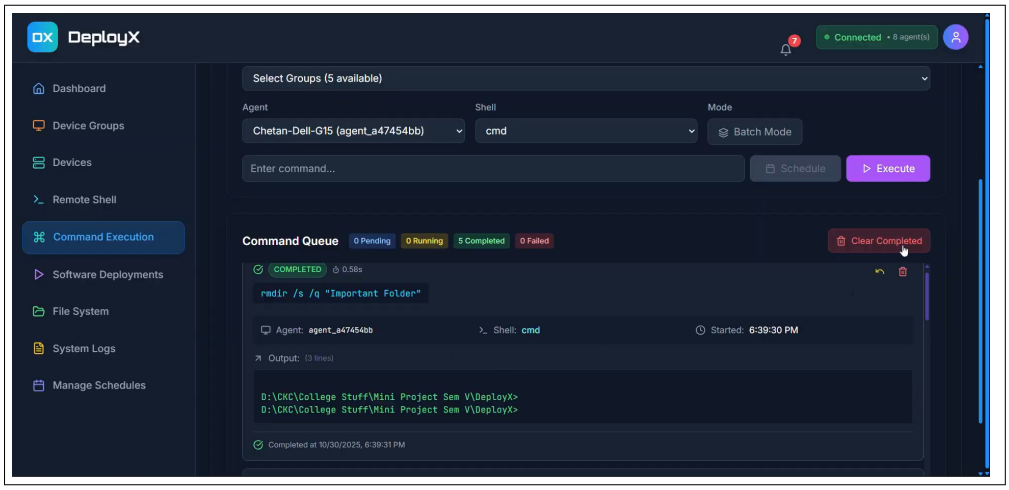

Figure 4.9: Command Execution Status

This screen displays the real-time Command Queue interface within the Command Execution module, where executed commands and their results are listed for monitoring and verification. The command queue provides detailed insight into command processing, including execution time, target agent, shell type, and status indicators such as **Completed**, **Failed**, **Pending**, or **Running**. Each command entry expands to show the executed command along with live terminal output returned from the remote machine. This allows administrators to

confirm successful execution or diagnose issues without needing separate access to logs or remote terminals.

The interface also includes management options such as **Clear Completed**, allowing users to remove finished tasks and keep the queue organized. Visual labels and icons enhance clarity by marking successful executions in green and failures in red. The queue enables sequential and batch execution tracking, supporting efficient automation workflows across multiple devices. Overall, this feature provides transparent visibility into remote operations, strengthens task auditing, and improves usability by centralizing execution history in a structured and readable format.

4.7 Software Deployments

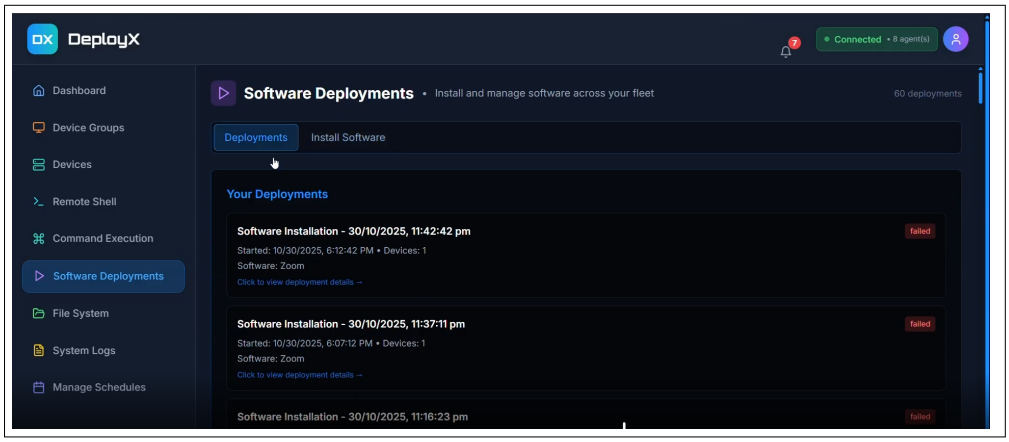

Figure 4.10: Software Deployment Overview

The Software Deployments interface allows administrators to manage and automate software installation across remote devices from a centralized dashboard. The screen includes two key sections: viewing existing deployments and initiating new software installations. The highlighted view shows the Deployments tab, where previously executed deployment tasks are listed along with details such as start time, target devices, software name, and execution status. Each deployment entry can be expanded to view additional information such as logs and error messages, enabling quick troubleshooting when failures occur. Status labels such as **Failed**, **Completed**, and **In Progress** help users track deployment results at a glance.

This module supports remote software rollout to individual devices or device groups, significantly reducing manual installation effort and improving operational efficiency in distributed environments. The interface includes controls for

reviewing deployment history and initiating new deployment tasks through the **Install Software** option. Notifications and real-time updates ensure administrators stay informed about ongoing operations and system performance. Overall, the Software Deployment feature enhances automation, saves time, and enables reliable management of application distribution across large networks.

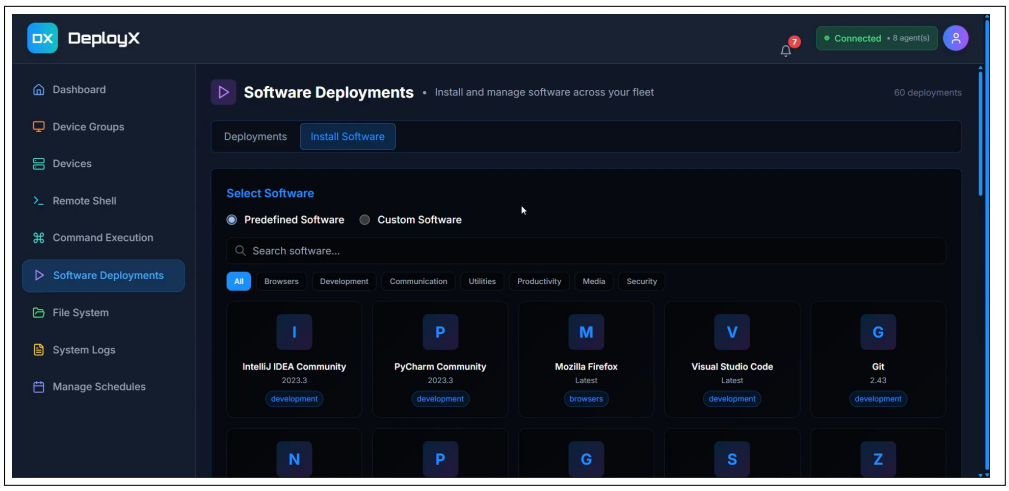

Figure 4.11: Install Software Interface

The displayed interface represents the Install Software section within the Software Deployments module of DeployX. This screen allows administrators to choose software packages to deploy remotely across selected devices or device groups. It provides two installation options — **Predefined Software**, which includes a curated list of commonly used applications such as IntelliJ IDEA, PyCharm, Visual Studio Code, Firefox, and Git; and **Custom Software**, enabling users to upload or specify installers for specialized needs.

The interface also includes a search bar and category filters such as **Browsers**, **Development**, **Utilities**, **Productivity**, and **Security**, making it easy to quickly locate required software. Each software item is displayed as a card showing the name, version, and category, allowing users to select one or more items for deployment. Once selected, administrators can assign the installation task to a device or group and initiate deployment without physically accessing the machines. This streamlined interface significantly reduces installation workload and improves scalability in environments such as colleges, computer laboratories, and enterprise networks. Overall, the Install Software screen enhances centralized automation and simplifies software management at scale.

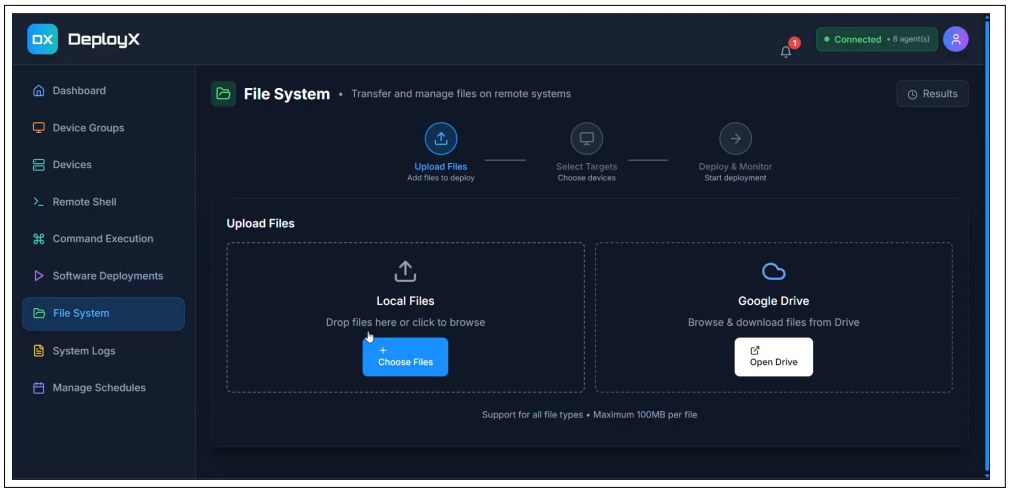

Figure 4.12: File System Interface

4.8 File System

The File System interface in DeployX enables users to transfer and manage files across remote machines in a streamlined and automated way. This screen allows administrators to upload files either from the local system or integrated cloud storage services such as Google Drive. Once files are selected, they can be deployed to chosen target devices and monitored in real time. The interface supports drag-and-drop uploads, browsing, viewing upload progress, and deploying files to multiple systems simultaneously. Designed for efficiency and simplicity, this module reduces the need for manual file transfer methods such as USB sharing or remote desktop copy-paste, making file distribution faster, more reliable, and scalable.

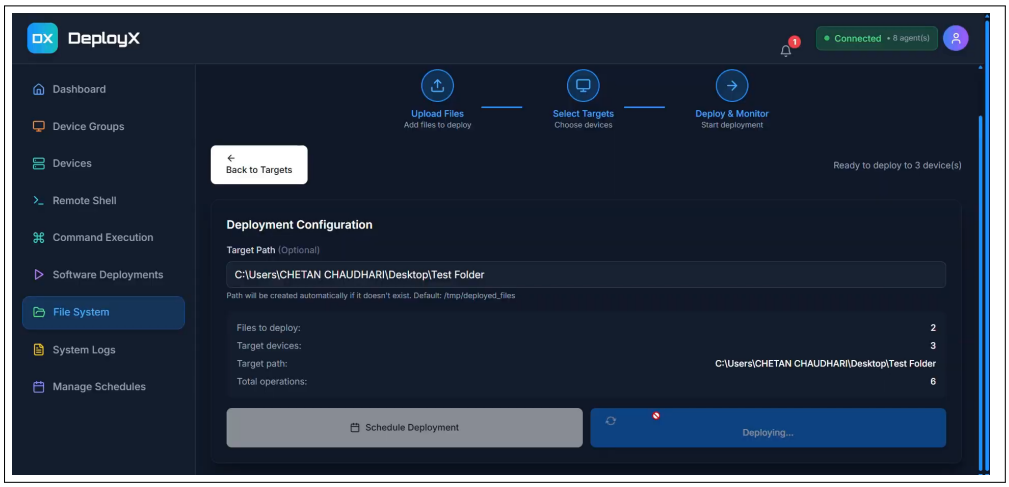

Figure 4.13: Deployment Configuration

The first image shows the Deployment Configuration stage within the File Sys-

tem module, where administrators configure the target path and deployment parameters before sending files to selected remote devices. This interface confirms which devices the files will be sent to and allows the user to specify the directory path where files should be stored. If the specified path does not exist, the system automatically creates it on the remote machine. The visual workflow at the top highlights the step-by-step deployment process: uploading files, selecting targets, and deploying while monitoring progress.

The configuration screen displays key summary information, including the count of files selected, number of target devices, and total operations to be executed. Users can also choose to schedule deployments for later execution, improving automation capabilities for organizations running routine deployments. Once configuration is complete, pressing the **Deploy** button initiates real-time deployment, with status updates provided continuously. This interface enhances reliability and control by allowing users to verify configuration details before executing critical file transfers.

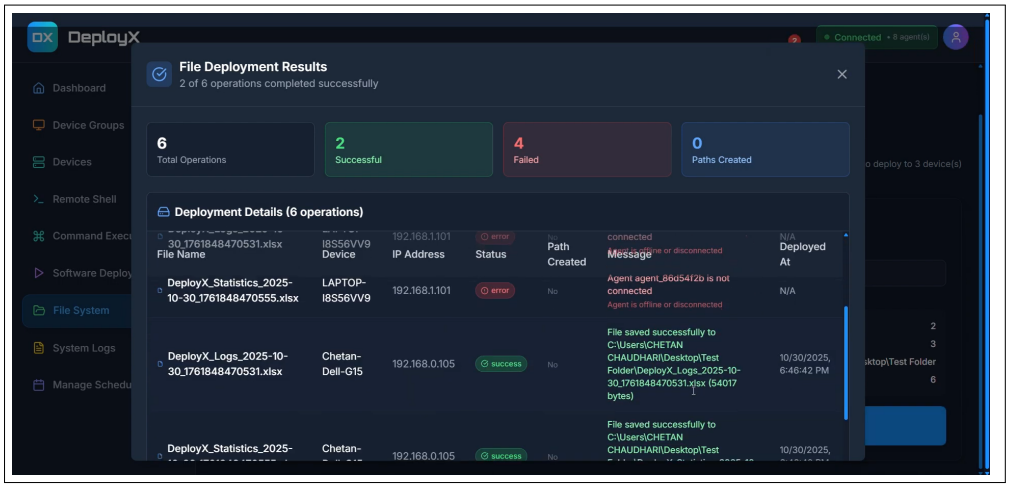

Figure 4.14: Deployment Result Summary

The second image shows the File Deployment Results window, which appears after the deployment process has finished. It provides a detailed summary of the deployment outcome, including the total number of operations performed and the count of successful and failed transfers. Each operation is listed with information such as file name, device name, IP address, deployment status, and response message. Status indicators such as success (green) and failure (red) help users instantly identify which deployments completed successfully and which require attention.

In the example shown, the interface indicates that only 2 out of 6 operations

were successful, while the remaining failed due to offline or disconnected devices. Detailed error messages enable troubleshooting without navigating through logs manually. This transparency ensures reliability, accountability, and traceability in large-scale distributed file transfer processes. Overall, the results screen provides clear post-deployment insights, allowing administrators to evaluate performance, retry failed operations, or refine targets accordingly.

4.9 System Logs

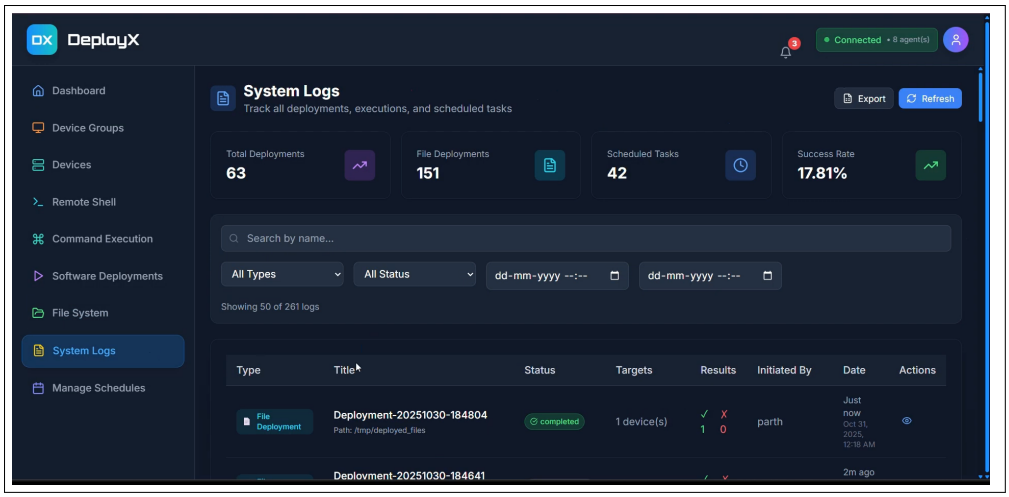

Figure 4.15: System Logs Overview

The System Logs screen displays a centralized record of all remote operations executed within DeployX, including deployments, command executions, file transfers, and scheduled tasks. At the top of the interface, key statistical summaries such as total deployments, file deployment count, number of scheduled tasks, and overall success rate allow administrators to analyze system performance and operational efficiency. The interface also includes a search bar and multiple filtering options (type, status, and date range), making it easy to locate specific records in large-scale environments. Additionally, the screen supports exporting logs for auditing or reporting purposes, and a refresh button ensures that the data displayed reflects the most recent system activity.

Below the summary statistics, a detailed log table lists individual operations along with attributes such as type, title, status, target devices, result counts, user who initiated the action, timestamp, and quick access icons for additional details. Status indicators such as **Completed**, **Running**, and **Failed** make it easy to visually interpret outcomes and identify areas requiring investigation.

This transparency enhances accountability and helps IT administrators maintain visibility and compliance across remote operations. Overall, the System Logs module strengthens monitoring and traceability within distributed control workflows.

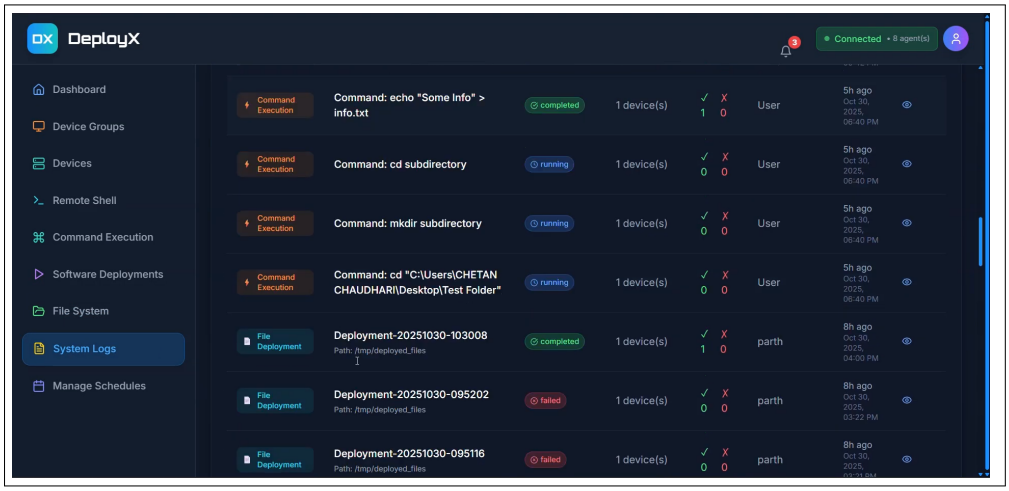

Figure 4.16: Detailed Log View

The second image provides an expanded view of individual log entries within the System Logs module, showing detailed records of command executions and file deployment operations. Each entry clearly displays the action performed, the command issued or file deployed, the number of affected devices, and the resulting success or failure status. Status badges such as **Completed**, **Running**, and **Failed** offer real-time clarity, enabling users to quickly check progress or identify problems without opening separate windows. Icons for viewing detailed results allow administrators to inspect logs further when debugging issues.

The detailed log listing also includes information about which user executed the operation and the timestamp of execution, supporting auditing, accountability, and historical tracking. By presenting logs in a structured and color-coded format, the interface improves clarity, speeds troubleshooting, and enables transparent post-operation verification. Overall, the detailed log view plays an essential role in evaluating performance, diagnosing failures, and maintaining reliability within DeployX.

4.10 Manage Schedule

The Manage Schedules screen allows administrators to view, organize, and control all automated scheduled tasks configured within DeployX. It displays a

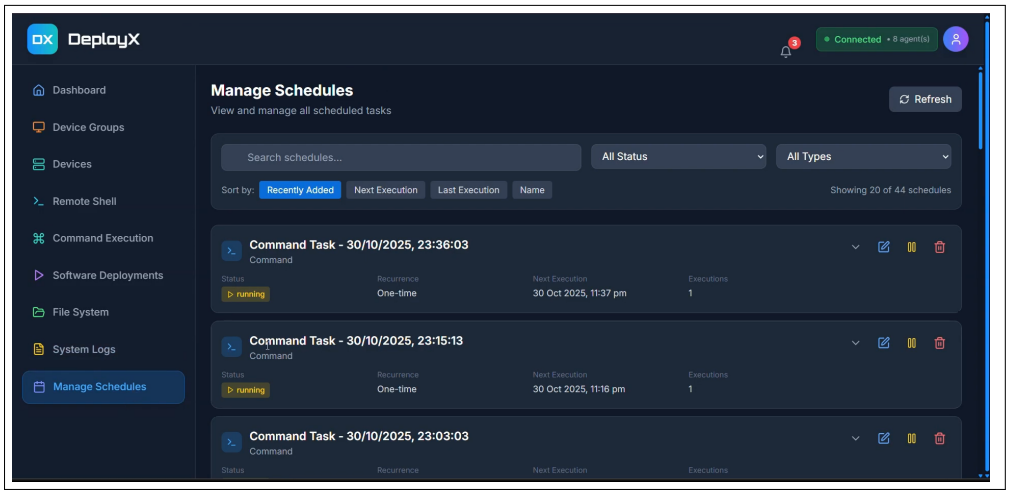

Figure 4.17: Manage Schedules Screen

searchable and filterable list of scheduled operations, including automated command executions and file deployment tasks. Sorting options such as **Recently Added**, **Next Execution**, **Last Execution**, and **Name** help users quickly navigate through large numbers of schedules. Each task entry clearly shows important execution details including status, recurrence type, next execution time, number of executions completed, and control icons for editing, pausing, resuming, or deleting tasks.

This central management interface enhances operational visibility and ensures that scheduled actions can be monitored and adjusted without manual reconfiguration. It helps IT administrators automate repetitive system management tasks such as system information retrieval, cleanup processes, periodic deployments, and health reporting. By eliminating the need for manual task oversight, this module supports improved efficiency, reliability, and proactive system maintenance across multiple remote systems.

The Edit Schedule interface enables users to modify the configuration details of an existing scheduled task. It provides fields for updating the task name, adjusting scheduled execution time, and modifying the command that will run automatically. The screen includes an intuitive time picker, allowing administrators to set precise local execution times, which are automatically converted into UTC to ensure server-level consistency across different time zones. This improves scheduling reliability and prevents execution conflicts.

Additionally, the interface includes a device selection panel, enabling users to specify which devices will execute the scheduled command. Administrators can update selections based on evolving operational requirements or device avail-

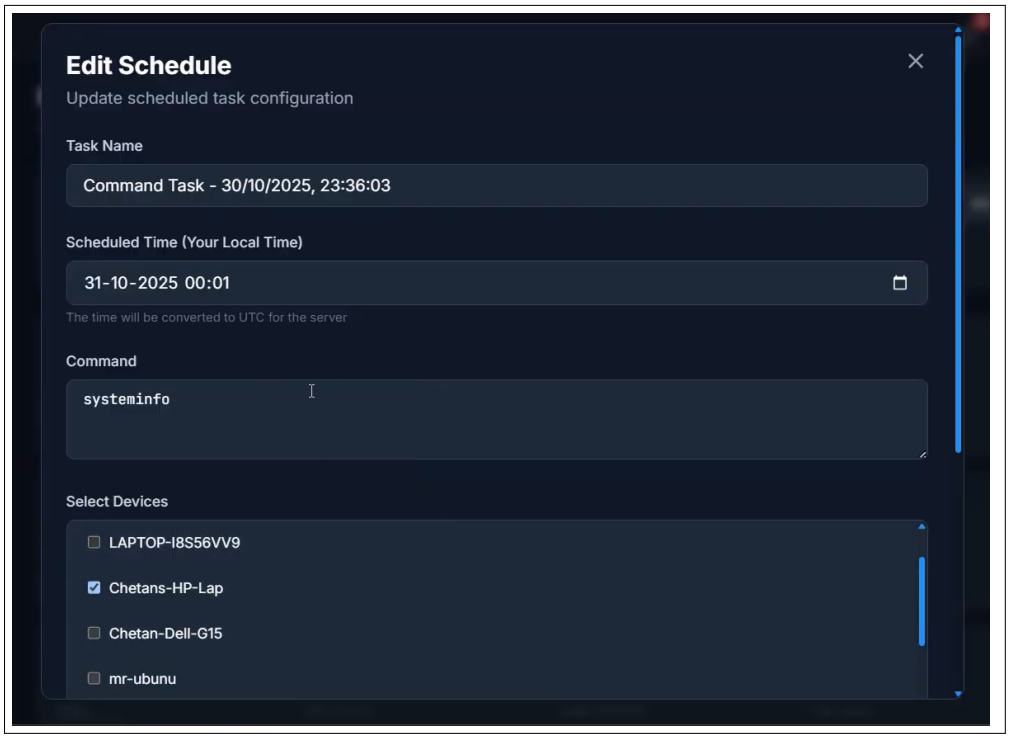

Figure 4.18: Edit Schedule

ability. Once modifications are saved, the task continues to execute automatically without further manual intervention. This capability improves flexibility and adaptability within automated workflows and strengthens reliability in distributed remote management environments.

Chapter 5

Results and Discussion

This chapter presents the intermediate results, testing observations, and overall analysis carried out during the development of DeployX. It highlights key module-level outcomes, final system performance, and comparisons with existing remote management and deployment tools.

5.1 Intermediate Results and Analysis

During the development of DeployX, major components were individually developed and tested to ensure stable integration and real-time performance. The interactive dashboard, live terminal execution, command queue engine, file transfer system, and software deployment mechanisms all functioned reliably during staging. Secure WebSocket communication was evaluated extensively to validate instant streaming between backend and agents. Real-time system statistics, agent heartbeat monitoring, and device grouping worked consistently across Windows, Linux, and macOS systems. Authentication, role-based access, and logging mechanisms were also tested to confirm data consistency and secure access handling.

Overall testing showed that DeployX's distributed architecture delivered efficient real-time control and minimized latency during remote operations. Agent connectivity remained stable even under heavy interaction such as continuous command streaming and large file transfers. These intermediate results con-

firmed the system's scalability, cross-platform compatibility, and readiness for real-world deployment across networks ranging from small labs to enterprise fleets.

5.1.1 Live Interactive Remote Shell

Intermediate Result: The Remote Shell module successfully provided real-time terminal access to remote machines through WebSocket communication. Users could execute system commands through CMD, Bash, or PowerShell based on OS type. Command outputs were streamed live without refresh requirements, and session management allowed selective agent switching.The Remote Shell module successfully provided real-time terminal access to remote machines through WebSocket communication. Users could execute system commands through CMD, Bash, or PowerShell based on OS type. Command outputs were streamed live without refresh requirements, and session management allowed selective agent switching.

Analysis: Testing confirmed instant command response with minimal latency and reliable output streaming. The feature eliminated the need for remote desktop tools for command-level control. Sessions could be safely terminated to prevent unauthorized access, ensuring both performance and security.

5.1.2 Command Execution System

Intermediate Result: The batch command execution system enabled the deployment of commands to multiple devices simultaneously. Commands were queued, executed, and tracked with status indicators such as pending, running, completed, and failed.

Analysis: The command queue operated accurately and consistently displayed results per device. This allowed large-scale automation of tasks like cleanup scripts, configuration updates, or diagnostics. The feature replicated real enterprise-grade management tools and significantly reduced manual workloads.

5.1.3 Software Deployment Module

Intermediate Result: DeployX supported predefined and custom software installation packages. The system enabled deployment to single or multiple devices with tracking and logging visibility.

Analysis: Software installation worked effectively during testing and provided useful feedback about deployment failure causes such as offline devices or unsupported formats. This feature demonstrated real-world applicability for labs and enterprises needing centralized software rollout.

5.1.4 File Transfer Management

Intermediate Result: The file system allowed uploads from local storage or Google Drive and deployed files to remote agent directories with monitoring feedback.

Analysis: File transfers were fast and stable, with partial-transfer handling for offline devices. Real-time progress visibility improved usability and troubleshooting.

5.1.5 System Logs and Activity Tracking

Intermediate Result: All operations—including commands, deployments, shell sessions, file transfers, and schedules—were logged in a structured format. **Analysis:** Logs helped track operational history and improve accountability. Export and filtering supported analytical review and performance benchmarking.

5.1.6 Scheduling and Automation

Intermediate Result: Automated tasks were scheduled and executed based on user-defined time, recurrence, and device selection.

Analysis: Scheduling eliminated repeated manual execution and improved operational efficiency. Scheduling tests showed accurate execution timing and reliability.

5.1.7 Integration Tests and Team Coordination

Intermediate Result: All independently developed modules were gradually merged using GitHub. Team communication was maintained via WhatsApp and regular Google Meet check-ins. Roles were clearly distributed: frontend (Chetan), backend/API (Parth), database (Nischay).

Analysis: Integration was smooth due to modular coding practices. No major merge conflicts or logical clashes occurred. Team stayed on schedule,

adapted quickly to issues, and maintained a unified vision. Mentor feedback was incorporated efficiently after each review round.

5.2 Final Results and Analysis

5.2.1 System Performance:

DeployX demonstrated high responsiveness and consistency across all modules in the final integrated environment. The dashboard loaded real-time device status and system metrics with a smooth UI experience. Remote command execution and terminal streaming performed with near-instantaneous response. File and software deployment operations completed efficiently even across multiple devices. Communication between agents and backend through WebSockets handled concurrency reliably without performance degradation. Throughout testing, the frontend interface remained consistently responsive and accessible across different device types and browsers.

5.2.2 System Accuracy:

DeployX maintained high operational accuracy in device state reporting, command execution results, system statistics, and file transfer logs. Device status updates (online/offline) were consistently synchronized with heartbeat signals. Output logs aligned precisely with remote system responses, and deployment results accurately reflected successful or failed operations. Logging supported reliable historical traceability and error tracking, and no inconsistency or data loss was recorded during tests.

5.3 Comparison with Existing Applications

5.3.1 Feature Comparison

Compared to traditional remote management platforms such as TeamViewer, AnyDesk, PDQ Deploy, or LANDesk, DeployX provides a more automation-centric and developer-friendly workflow. Unlike common tools that rely on remote screen control or manual access, DeployX focuses on real-time command-level control, parallel execution, fleet-based software deployment, and automated file transfer. While many existing platforms do not provide per-device execution logs, batch execution, or interactive shell streaming, DeployX integrates all of these features through a unified dashboard. Scheduling automation,

agent-based backend execution, and integrated system activity analytics further differentiate it from typical tools used in labs and enterprise settings.

5.3.2 Performance Comparison

DeployX outperforms legacy remote access tools in execution speed and automation flexibility due to its WebSocket-based real-time pipeline instead of screen-capture streaming. Remote shell execution and command queues respond significantly faster than remote-desktop–based approaches. File transfer and software deployment operations complete with detailed progress tracking rather than blind batch execution. System logs and fully automated scheduling improve operational transparency and efficiency, which is missing in most competing solutions. Overall, DeployX delivers higher automation, reliability, scalability, and performance in distributed computer management environments.

5.4 Summary

DeployX is a robust MERN-based remote control and automation platform designed to manage and operate large fleets of distributed computers in real time. Through interactive remote shells, batch command execution, software deployment, file transfer, and scheduling automation, DeployX reduces manual workload and enhances operational control. The system proved reliable, secure, and performance-oriented during testing, demonstrating high accuracy and strong real-time responsiveness. Compared to existing remote management tools, DeployX delivers improved automation features, faster execution pipelines, centralized monitoring, and detailed logging, making it an efficient and scalable solution for academic labs, corporate IT departments, and enterprise device management environments.

Chapter 6

Conclusion and Future Work

This chapter summarizes the key findings and outcomes from the development, testing, and execution of DeployX. It also reflects on project management learnings and outlines potential areas for future enhancement that can be built upon the existing architecture.

6.1 Conclusion

The DeployX project successfully delivers a robust and scalable solution for centralised remote system control, command execution, software deployment, and file management across distributed computer networks. By integrating lightweight Python agents with a powerful FastAPI-based backend and an interactive React.js dashboard, DeployX bridges the operational gap between manual remote administration and automated fleet management. The platform meets its core objectives by enabling secure real-time command execution, efficient multi-device control, centralized monitoring, and automation—all through a modern and intuitive web interface.

Key Outcomes and Highlights:

- **Real-Time Command Execution:** Commands executed instantly across multiple systems with live output streaming via Socket.IO.
- **Interactive Dashboards:** Real-time status visibility and device monitor-

ing for connected systems.

- **Batch and Group Execution:** Execute commands or deploy software across groups of computers simultaneously.
- **Secure Remote Access**: Token-based agent authentication, encrypted communication, and permissioncontrolled actions.
- **Software Deployment System:** Install predefined or custom software packages across selected machines in parallel.
- **Centralized File Sharing:** upload, distribute and manage files across remote System.
- **System Logs and Scheduling:** Automatically track activities and schedule recurring tasks.

In conclusion, DeployX proves to be an efficient and scalable remote management platform capable of handling real-world enterprise requirements. Its modular architecture, modern technology stack, and real-time capability establish it as a next-generation tool for distributed IT infrastructure control and automation.

6.2 Lessons Learned from Project Management

The development of DeployX provided rich learning experiences in end-to-end full-stack development, network communication, and scalable system design. Beyond technical growth, the project enhanced problem-solving, collaboration, and time-planning abilities essential for real-world engineering.

Key Lessons Gained:

- **Full-Stack Engineering** Hands-on experience with a production-style MERN stack, including FastAPI, WebSockets, and PostgreSQL.
- **Real-Time Communication:** Deep understanding of Socket.IO for bi-directional communication and live streaming of outputs.
- **Agent-Based Architecture Design:** Building lightweight Python agents capable of executing commands securely via subprocess pipelines.
- **Security Considerations:** Implementing authentication, identity validation, logging, and action tracking for safe remote operations.

- **Team Coordination and Version Control:** Smooth collaboration through GitHub, frequent synchronization, and modular development.
- **Testing and Debugging Workflow:** Iterative validation of command execution, deployment logic, and agent-server communication reliability.

Through this project, we not only developed a functional tool but also honed the skills required to design and deploy end-to-end solutions. The practical challenges helped us grow technically and adaptively, preparing us for more complex software development tasks in the future.This project strengthened both technical and managerial capabilities, preparing the team for future industry-level system engineering work.

6.2.1 Requirement Gathering and Understanding

The development of DeployX began with a clear objective of addressing real-world challenges in remote computer management, automation, and large-scale device orchestration within enterprise, institutional, and laboratory environments. Traditional remote-access solutions lacked automation, batch command execution, centralized monitoring, and real-time visibility, which motivated the requirement analysis phase. To ensure DeployX was practical, user-centric, and scalable, extensive effort was dedicated to gathering, validating, and refining user requirements through analytical and research-driven methods.

Key Requirement Gathering activities:

- **Stakeholder Discussions:** Conducted discussions with IT administrators, lab managers, system engineers, and technical staff to understand current pain points in managing distributed systems and software deployment workflows.
- **Problem Observation:** Observed repetitive manual tasks such as updating software individually across machines, command execution through remote desktops, slow troubleshooting, and lack of visibility into system health.
- **Requirement Documentation:** Documented functional requirements including real-time remote shell, batch execution, software deployment, file transfer, device grouping, monitoring dashboards, and automated scheduling. Non-functional requirements included security, reliability, low latency, scalability, and cross-platform agent support.

- **Use-Case Analysis:** Identified multiple user roles (System Administrator, Operator, and Technician) and mapped them to execution privileges, dashboards, automation capabilities, and device management workflows.
- **Technical Feasibility Checks:** Evaluated feasibility of features such as WebSocket-based real-time command streaming, Python subprocess execution on agents, file deployment handling, multi-device scheduling, and performance-aware FastAPI integration.

This structured requirement study ensured that DeployX was designed not merely as a remote access tool but as an intelligent automation and monitoring platform capable of enterprise-level deployment and future scalability.

6.2.2 Time Management

Time management played a critical role in the successful execution of the DeployX project. Development followed a structured milestone-based workflow starting with requirement analysis, technology stack selection, architecture planning, prototype implementation, UI/UX development, backend integration, agent development, testing, and final optimization. A weekly planning cycle was followed to ensure consistent progress across all modules.

The timeline included dedicated phases for researching remote execution technologies, learning communication protocols such as WebSockets and WebRTC, and performance testing under different network environments. Regular review sessions and internal testing cycles helped identify issues early and refine the system iteratively. Parallel development of frontend, backend, and agent components ensured maximum efficiency and minimal roadblocks. Structured task segmentation and continuous progress tracking helped maintain balance between academic deadlines and development quality.

6.2.3 Collaboration and Communication

Collaboration and communication were essential components in developing DeployX, enabling smooth coordination across different development roles and technical responsibilities. Tasks were divided into specialized areas such as UI/UX design, backend API development, agent architecture, WebSocket pipeline implementation, and database management. This modular approach allowed the team to work independently while maintaining alignment through shared objectives.

Team interaction was maintained through regular virtual meetings, continuous updates on WhatsApp and Discord, and collaborative version control using GitHub for managing code changes and ensuring development transparency. Shared documentation, architecture diagrams, and status tracking tools supported clear communication and planning. Constructive feedback from mentors and peers helped improve system design decisions and feature prioritization. This collaborative workflow not only ensured a stable development path but also improved problem-solving, technical decision-making, and teamwork discipline.

6.3 Future Work

DeployX has a solid foundation for expansion, and multiple enhancements can substantially increase usability, automation intelligence, and enterprise scale.

6.3.1 AI-Based Predictive Analytics :

Introduce ML models to detect risky command patterns, anticipate system failures, and recommend preventive actions.

6.3.2 Microservices Deployment and Cloud Hosting:

Convert backend into microservices, deploy via Kubernetes, and integrate CI/CD pipelines for dynamic scaling and container orchestration.

6.3.3 Mobile and Voice Command Support:

Build Android/iOS administration apps and integrate voice assistants (e.g., Google Assistant/Jarvis style voice execution).

6.3.4 Integration with DevOps Tools:

Add support for tools like Jenkins, Ansible, Azure DevOps, GitHub Actions for automated environment provisioning.

6.3.5 Cross-Platform Remote Desktop:

Add remote screen view and remote desktop control similar to AnyDesk or TeamViewer.

6.3.6 Plugin Marketplace:

Allow third-party extensions for custom automation scripts, security tools, and hardware management plugins.

6.3.7 SNMP and IoT Integration:

Monitor and manage routers, switches, and IoT devices for end-to-end enterprise infrastructure visibility.

6.4 Summary

DeployX is a powerful full-stack system designed to automate remote computer management across LAN and Internet-based environments. Through real-time command execution, software deployment, file distribution, scheduling automation, and centralized logging, it significantly reduces manual effort and enhances operational productivity. The project demonstrates how modern web technologies, socket-based communication, and lightweight agents can combine to build scalable and intelligent remote administration ecosystems. DeployX's architecture supports future enhancements, making it a strong candidate for enterprise-level adoption and continued research.

References

[1] TeamViewer – Remote Access and Remote Control Software: https://www.teamviewer.com/en/

[2] Chart.js: https://www.chartjs.org/

[3] ReportLab: https://www.reportlab.com/

[4] DeployX Product Concept Reference Documentation: https://socket.io/docs/v4/

[5] PostgreSQL: https://www.postgresql.org/docs/

[6] FAST API Video Tutorial: https://www.youtube.com/watch?v=rHux0gMZ3Eg

GitHub Repository

https://github.com/Nischay-loq/DeployX.git

Acknowledgements

We, the undersigned, would like to extend our sincere gratitude to everyone who supported us throughout the completion of our college mini project. First and foremost, we are deeply thankful to our project guide Dr. Vaishali Kavathekar, for her expert guidance, constructive feedback, and constant encouragement. Their insights into software development and practical knowledge helped us navigate the challenges we encountered during the development of this billing and inventory management system.

We would also like to thank our college for providing us with the resources and opportunity to work on this project. The experience has enhanced our understanding of software design, teamwork, and project management. Finally, we are immensely grateful to our families and friends for their unwavering support, understanding, and motivation throughout this project.

(————————————-)
Chetan Chaudhari - 10

(————————————-)
Nischay Chavan - 14

(————————————-)
Parth Shikhare - 53

Date: