RSET
RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Software Requirement Specification

On

Terminal Velocity

By

Eesha Ajith (U2203086)
Elena Thomas Kochupurakal (U2203087)
Giribala Arun (U2203099)
Maysa Sameer (U2203139)

Under the guidance of

Mr. Sandy Joseph

Department of Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)
Rajagiri Valley, Kakkanad, Kochi, 682039
February 2025

Software Requirements Specification

for

Terminal Velocity

Version 1.0 approved

Prepared by

U2203086 Eesha Ajith

U2203087 Elena Thomas Kochupurakal

U2203099 Giribala Arun

U2203139 Maysa Sameer

Guided By: *Mr. Sandy Joseph, Asst. Professor, RSET*

Table of Contents

1. Introduction

1.1 Purpose

This document outlines the software requirements for a web-based Linux learning platform, designed to teach essential Linux commands through interactive tutorials and challenges. The platform aims to provide an engaging, gamified learning experience that enhances users' Linux command skills in an enjoyable and immersive way.

1.2 Product Scope

Terminal Velocity is designed to provide an interactive and gamified approach to learning Linux command skills. Its primary goal is to guide users through a series of tutorials and challenges that progressively build their proficiency with essential Linux commands. By combining educational content with a game-like structure, the platform aims to make learning both fun and effective.

A key objective of the game is to help users overcome the common intimidation that often accompanies their first experience with a Linux system. Many new users find the command line overwhelming, and this platform addresses that challenge by providing a learning environment that breaks down complex concepts into bite-sized tasks.

To keep learners motivated, the system incorporates a badge-based reward system. As users complete challenges and master new commands, they earn badges that recognize their achievements. This gamified approach not only rewards progress but also encourages users to keep advancing through increasingly challenging tasks, making the learning process both engaging and rewarding.

2. Overall Description

2.1 Product Perspective

The Linux learning game is a standalone web application created to help users develop and master Linux command skills in an interactive and engaging way. It features a comprehensive set of tutorials and challenges that guide users from beginner-level basics to more advanced concepts and techniques, with more focus on task-based lessons. Each tutorial is designed to provide clear explanations, and opportunities for hands-on trials. The application also incorporates a controlled, web-based environment where users can execute real Linux commands, enabling them to practice and refine their skills in a safe, simulated terminal. By combining learning materials with practical exercises, the platform ensures an effective and immersive educational experience for users at all proficiency levels. As shown in Figure 1.1, the system architecture consists of several key components that work together to provide an interactive and immersive learning experience for users:

- Frontend: The user interface is the front-facing component that allows users to interact with the application.
- Backend: The backend manages the logic for user authentication, tutorial content delivery, challenge progression, and command execution.

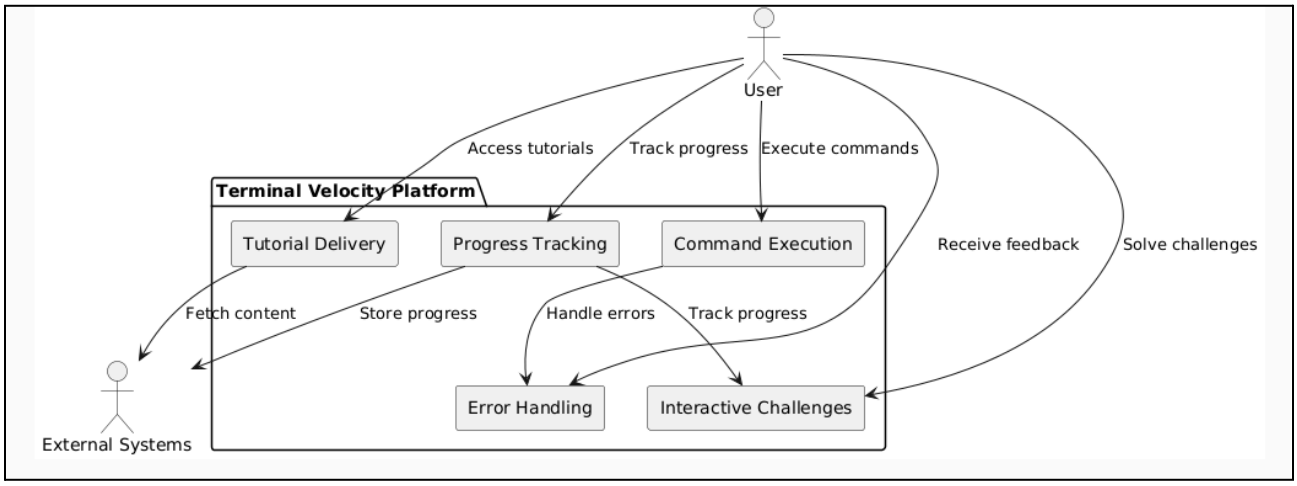

Figure 2.1: System Architecture Diagram

2.2 Product Functions

2.2.1 Tutorial Delivery: Provide structured lessons covering Linux command-line skills, progressing from basic to advanced levels.

2.2.2 Interactive Challenges: Present users with tasks and problem-solving exercises to test and reinforce their understanding of Linux commands.

2.2.3 Simulating a Linux Command-Line Interface: Users can input Linux commands directly into a simulated terminal window and receive outputs just like they would in a real Linux terminal.

2.2.4 Command Execution: Enable users to practice real Linux commands in a simulated, web-based terminal environment.

2.2.5 Progress Tracking: Monitor and display user progress through tutorials and challenges, allowing for a clear learning path.

2.2.6 Feedback Mechanism: Offer immediate feedback on user actions, including error explanations and suggestions for improvement.

2.2.7 User Management: Allow users to create and manage accounts to personalize their learning experience and save progress.

As shown in Figure 2.2, the UML diagram illustrates the key components of the system and their interactions.

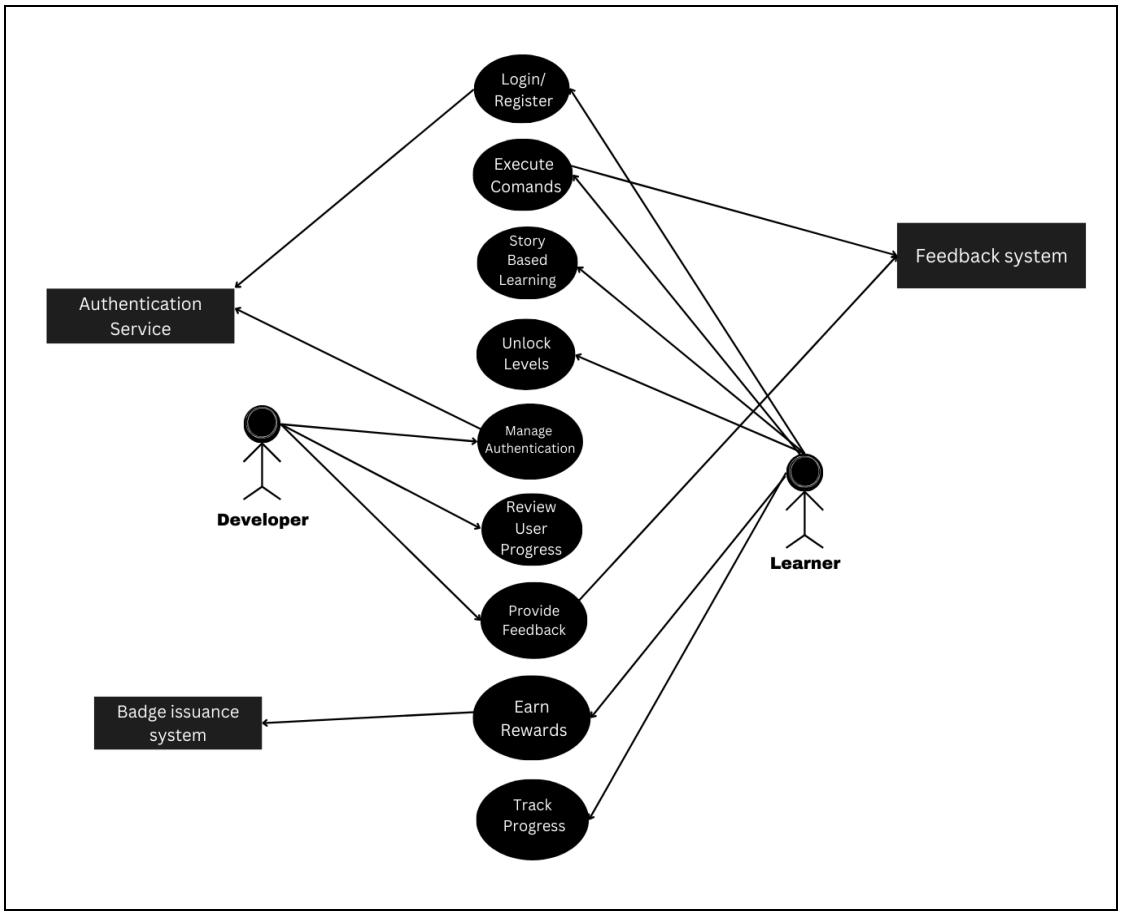

Figure 2.2: UML Diagram

2.3 Operating Environment

The Linux learning game will operate within the following environment:

Hardware Platform

1) **Client-Side**: A standard laptop or desktop computer with basic computing capabilities, including at least 4GB RAM, a modern processor capable of running a web browser efficiently and internet connection for accessing online features.
2) **Server-Side**: A standard server setup or cloud instance with sufficient resources (e.g., 2 vCPUs, 8GB RAM, and 50GB storage) to host the application.

Operating Systems

1) **Client-Side**: Compatible with commonly used operating systems such as:
    a) Windows 10 and later
    b) macOS 10.14 and later
    c) Linux distributions (e.g., Ubuntu, Fedora)
2) **Server-Side**: Runs on a Linux-based server operating system, such as Ubuntu 20.04 LTS or equivalent, optimized for hosting web applications.

2.4 Design and Implementation Constraints

2.4.1. Limited Backend Processing Power:

a) Due to the restricted nature of the web application, the backend should be optimized for basic operations. Heavy computations, complex algorithms, or high-frequency requests must be avoided to ensure the system runs efficiently within limited resources.

2.4.2. Basic User Authentication:

a) The authentication system is required to be simple and must rely on Firebase's built-in authentication mechanisms. Developers will have limited flexibility to customize or extend authentication features beyond what Firebase offers out-of-the-box.

2.4.3. Limited Database Structure:

a) The database should be designed for lightweight, minimal storage and querying. Developers will need to optimize data models for basic tracking of user progress, and badges. Complex relational database queries or large data structures may not be feasible.

2.4.4. No Real-Time Collaboration:

a) The system does not support features that enable real-time collaboration or multiplayer functionality due to backend and frontend limitations. User interactions are isolated, and any need for simultaneous interaction between users must be avoided.

2.4.5. Cross-Browser Compatibility:

a) The game must be designed for compatibility with modern web browsers, but the scope is limited to basic HTML, Tailwind CSS, and JavaScript. Advanced browser-specific features should be avoided to maintain simplicity and cross-browser consistency.

2.4.6. Basic Level Security Practices:

a) The application must adhere to basic security best practices, such as input validation, data encryption, and secure token storage for Firebase authentication. However, there will be no complex security features like multi-factor authentication, user role management.

2.5 Assumptions and Dependencies

2.5.1 Assumptions:

a) **Availability of Firebase Services:** It is assumed that Firebase services (Authentication and Database) will remain available and accessible for the project. Any changes or outages in Firebase's offerings, pricing models, or availability could affect the project's implementation.

b) **Internet Access for Users:** The game assumes users will have consistent internet access. If there are scenarios where users have intermittent or no internet, the game's functionality (especially involving authentication or game progress) could be limited.

c) **Browser Compatibility:** It is assumed that users will use modern web browsers (e.g., Google Chrome, Mozilla Firefox, Safari), and that all browser updates and standards will align with the features implemented in the project. Issues may arise if users are using outdated browsers or browsers with incomplete support for HTML5, Tailwind CSS, and JavaScript.

d) **No Complex Infrastructure:** The project assumes that there will be no requirement for a complex server infrastructure (e.g., load balancing, multiple servers). If the system scales beyond a basic number of users, there could be performance bottlenecks.

2.5.2 Dependencies:

a) **Firebase:** The project is dependent on Firebase for user authentication and database services. Any updates or service disruptions in Firebase could directly affect the project's backend functionality.

b) **Frontend Libraries and Frameworks:** The game depends on basic frontend technologies (HTML, Tailwind CSS, JavaScript). Changes in these libraries' versions or deprecated features could affect the development process.

c) **User's Device Performance:** The performance of the game depends on the hardware and browser capabilities of users' devices. If users are on lower-end devices, there could be limitations in rendering the UI or executing the simulated terminal environment.

d) **Security Compliance:** If the project needs to comply with specific security standards (e.g., GDPR, CCPA) for data storage or user authentication, this could introduce additional requirements for the backend and database structure, affecting the scope and design of the project.

e) **Browser Updates:** Assumption is made that major browser updates will not introduce breaking changes to the core web technologies used in the project. If browsers make changes that deprecate essential features, the system may need rework or fixes.

3. External Interface Requirements

3.1 User Interfaces

The UI will adhere to modern GUI standards, ensuring consistency with clear navigation, labeled menus, and standard action buttons (e.g., "Submit," "Cancel"). Error messages will follow a standardized format with error codes and corrective actions. A responsive design will ensure compatibility across desktop, laptop, and tablet devices, adjusting to different screen sizes and input methods. UI specifications, including layout and components, will be documented separately for detailed design.

3.2 Hardware Interfaces

The application will support devices with modern web browsers (desktop, laptop, tablet) using standard input methods such as keyboard, mouse, or touch. It will employ a responsive web design that adapts to different screen sizes, resolutions, and input methods. No specialized hardware is required, and communication will occur via standard web protocols (HTTPS) for secure data transfer between the client and server.

3.3 Software Interfaces

This web application is locally hosted and does not rely on third-party libraries, external databases, or secure communication protocols since no internet connection or remote services are involved. As such, the application operates in a contained environment on a local machine. The key software components include the frontend, which is built using HTML, Tailwind CSS, and JavaScript, and the backend, where the server-side logic is implemented using JavaScript to handle user input and execute necessary operations. Firebase is used solely for storing user progress, and data is handled locally without any external API calls. Since there is no data sharing or communication with other software components, there are no specific protocols or data formats that need to be adhered to. The application does not rely on secure communication (such as HTTPS) as it operates entirely within the local environment. Additionally, since no external

dependencies are involved, there are no complex data exchange protocols or services to define.

3.4 Communications Interfaces

The application will use HTTPS (TLS/SSL) for secure client-server communication, with data transmitted using HTTP methods (GET, POST, PUT, DELETE). JSON will be used for data exchange between the frontend and backend. All communication will follow industry-standard protocols for security and efficiency.

4. System Features

4.1 Interactive Learning Environment

4.1.1 Description and Priority

The platform offers a simulated terminal that allows users to practice Linux commands in a real-time environment, enhancing hands-on experience and practical knowledge. Priority: High.

a) **Benefit**: Enables interactive learning and builds practical Linux command proficiency.
b) **Penalty**: Could overwhelm users if the difficulty progression is not well-calibrated.
c) **Cost**: Development and maintenance of the adaptive system.
d) **Risk**: Misinterpretation of performance data may result in ineffective progression adjustments.

4.1.2 Stimulus/Response Sequences

a) **User Action 1:** User enters a Linux command in the simulated terminal.
System Response 1: The system processes the command and displays the output, providing a realistic experience.
b) **User Action 2:** User successfully completes a task or set of commands.
System Response 2: The system tracks performance and adjusts the next set of challenges, gradually increasing in difficulty.

4.1.3 Functional Requirements

a) **REQ-1:** The system must allow users to enter Linux commands into a simulated terminal and execute them in real-time.
b) **REQ-2:** The system should adapt the difficulty of tasks based on the user's performance and progress, ensuring a personalized learning experience.
c) **REQ-3:** The system must provide clear feedback and suggestions when a user inputs an incorrect command.

4.2 Gamified Progression System

4.2.1 Description and Priority

The platform incorporates a structured, gamified progression system that rewards users with achievements and badges as they complete tasks. Priority: High.

a) **Benefit**: Provides an engaging, motivational structure that drives continued learning.
b) **Penalty**: Risk of the user focusing too much on rewards rather than skill development.
c) **Cost**: Designing and implementing a complex leveling system.
d) **Risk**: The pacing of levels must be well-balanced to avoid frustration or disengagement.

4.2.2 Stimulus/Response Sequences

a) **User Action 1:** The user completes a set of tasks related to a specific functionality (e.g., File Management).
   System Response 1: The system rewards the user with badges or achievements based on their performance.
b) **User Action 2:** The user progresses through levels, completing challenges.
   System Response 2: The system unlocks new levels, each with more advanced tasks focusing on different Linux commands.

4.2.3 Functional Requirements

a) **REQ-1:** The system must track user progress through levels and unlock new challenges as users complete tasks.
b) **REQ-2:** The system should reward users with badges and achievements when mastering specific commands.
c) **REQ-3:** The system should progressively increase task difficulty and introduce new command topics in a structured way.

4.3 Terminal Simulator

4.3.1 Description and Priority

This feature simulates a Linux terminal where users can practice commands directly within a web-based interface. It allows for real-time feedback and command validation, enhancing the learning process. Users can track their progress and practice a variety of commands. Priority: High.

4.3.2 Stimulus/Response Sequences

a) **User Action 1:** The user selects a command challenge or tutorial in the application.

System Response 1: The system displays a simulated terminal interface.

b) **User Action 2:** The user types a Linux command into the terminal.

System Response 2: The system validates the entered command, and if correct, executes the command within the simulated environment, providing feedback.

c) **User Action 3:** The user receives real-time feedback, such as a success message or an error prompt.

System Response 3: If the command is correct, the system updates the user's progress and unlocks the next challenge. If the command is incorrect, the system provides hints or suggestions for correction.

4.3.3 Functional Requirements

a) **REQ-1:** The application must provide an interactive, simulated Linux terminal interface for the user to practice commands.

b) **REQ-2:** The application must validate the entered Linux commands and provide real-time feedback.

c) **REQ-3:** The application must offer hints or feedback for incorrect command usage.

d) **REQ-4:** The terminal simulator must execute the commands in a controlled, safe environment to prevent damage to the local system.

e) **REQ-5:** The application should allow users to reset the simulated environment for practice purposes without affecting their progress.

4.4 Performance Tracking and Feedback

4.4.1 Description and Priority

The system continuously tracks user performance, offering insights into executed commands, common mistakes, and overall progress. A dashboard displays detailed reports, helping users identify areas for improvement and monitor their learning journey effectively. Priority: High.

a) **Benefit:** Ensures users can track their learning and identify areas needing improvement.
b) **Penalty:** Users may become discouraged by frequent error reports.
c) **Cost:** Developing a comprehensive tracking and feedback system.
d) **Risk:** Feedback may become overwhelming if not well-structured.

4.4.2 Stimulus/Response Sequences

a) **User Action 1:** The user completes a series of commands or tasks.

   System Response 1: The system updates the progress dashboard, reflecting the user's performance.

b) **User Action 2:** The user views the performance dashboard.

   System Response 2: The system presents a detailed report on the user's executed commands, errors made, and suggestions for improvement.

c) **User Action 3:** The user repeatedly makes a specific error in executing commands.

   System Response 3: The system flags this as a common mistake and provides targeted suggestions to address it.

4.4.3 Functional Requirements

a) **REQ-1:** The system must record and display user performance metrics, including executed commands, errors, and mistakes.

b) **REQ-2:** The system should provide actionable feedback on mistakes and offer suggestions for improvement.
c) **REQ-3:** The system must include a progress dashboard to allow users to track their learning journey.

4.5 Error Handling

4.5.1 Description and Priority

The system provides clear, specific error messages when users input incorrect commands, accompanied by suggestions or hints to guide correction. This ensures users can learn from mistakes without becoming frustrated. Priority: High.

a) **Benefit:** Helps users improve by addressing errors in real-time.
b) **Penalty:** Misleading error messages could hinder learning.
c) **Cost:** Development of an intelligent error-handling system.
d) **Risk:** Inconsistent or unclear error feedback may confuse users.

4.5.2 Stimulus/Response Sequences

a) **User Action 1:** The user enters an invalid Linux command.

   System Response 1: The system displays an error message, explaining the mistake and suggesting a solution.
b) **User Action 2:** The user views the error message and follows the suggestion.

   System Response 2: The system allows the user to retry the command, providing immediate feedback on the new input.

4.5.3 Functional Requirements

a) **REQ-1:** The system must display a clear error message when a user enters an invalid command.
b) **REQ-2:** The system should provide helpful hints or suggestions to correct the error.
c) **REQ-3:** The system must allow users to retry commands after receiving feedback.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 User Authentication Speed

The authentication process should be completed within 2 seconds. This ensures a smooth and seamless experience for users without frustrating delays. Any delays in authentication may lead to a poor user experience and increased dropout rates.

5.1.2 Page Load Time

The application should load within 3 seconds on standard internet connections. This is critical to provide a responsive experience, keeping users engaged from the moment they access the game. A slow loading time may result in high bounce rates and dissatisfied users.

5.1.3 Database Query Performance

Database queries must be executed within 1 second. This ensures that users can quickly view their progress and rankings without waiting. Slow queries may result in a laggy user experience and hinder the game's overall responsiveness.

5.1.4 Frontend Rendering Efficiency

The user interface should render efficiently even on low-end devices. This requires ensuring the game is optimized for performance, with minimal resources required for visual elements. Poor rendering performance on older devices could alienate users with less powerful hardware.

5.2 Safety Requirements

5.2.1 Authentication and Access Control

The system must ensure secure user authentication through Firebase and prevent unauthorized access to user accounts. Insecure login attempts (e.g., brute-force attacks) should be detected, and users should be locked out after a certain number of failed attempts.

5.2.2 Error Handling and Feedback

The application must provide clear error messages that do not expose sensitive internal information. If an error occurs, the system should safely handle it without crashing or exposing the application to security risks. The error feedback system must guide users without revealing implementation details that could be exploited.

5.2.3 Input Validation and Command Restrictions

User inputs must be validated to prevent malicious commands or code injection that could harm the system or users. The simulated terminal must restrict potentially harmful Linux commands, such as system shutdown or file deletion commands, to ensure no damage occurs. All inputs should be sanitized to prevent security vulnerabilities.

5.3 Security Requirements

5.3.1 User Authentication and Authorization

All users must authenticate via Firebase Authentication, which requires a secure username and password combination. User identity verification should be completed before accessing any protected areas of the application or submitting any data.

5.3.2 Privacy Policy and User Consent

A clear and concise privacy policy must be provided to users at the point of account creation, outlining what data is collected, how it will be used, and how users can

manage or delete their data. The system must acquire explicit user consent before collecting any personal data.

5.3.3 Data Privacy Considerations

The application implements minimal authentication using local session management, without requiring external authentication services. All data is stored locally, eliminating risks of unauthorized access or data interception. Since no personal information is stored or shared beyond local usage, there are no external compliance requirements. Due to the absence of external dependencies or third-party access points, no additional security or privacy certifications are required.

5.4 Software Quality Attributes

5.4.1 Usability

The system must provide an intuitive and easy-to-navigate user interface. Users should be able to start the game and access key features within 3 clicks, ensuring an easy learning curve. The goal is to prioritize ease of use over ease of learning, minimizing the time required for users.

5.4.2 Reliability

The game must have an uptime of at least 99.59%, ensuring users can reliably access the platform for learning and practice. This will minimize downtime and ensure that users can trust the system for consistent learning experiences. Any failures or interruptions in service should be addressed within 24 hours.

5.4.3 Maintainability

The codebase should be modular and well-documented, allowing developers to easily update or modify features as needed. The system should be designed for

easy bug tracking and fixing, with the goal of reducing maintenance time by 30% after the initial release. This will allow for efficient updates without significant effort.

5.4.4 Flexibility

The system should be flexible enough to accommodate potential feature expansions, such as adding new challenges, tutorials, or additional user tracking systems. New features should be able to be integrated with minimal disruption to the existing functionality. This flexibility will help adapt to future needs without requiring a complete redesign.

6. Conclusion

In conclusion, this Software Requirements Specification (SRS) document outlines the essential features, functionalities, and system components of the Linux Learning Game. The platform is designed to provide an engaging, interactive, and structured environment for users to develop and master Linux command-line skills. By combining comprehensive tutorials, hands-on challenges, and a simulated terminal, the system ensures that users can progress from basic to advanced levels at their own pace.

The system is built with flexibility and scalability in mind, offering features such as user account management, progress tracking, and function-based levels, which enhance the learning experience. The seamless integration of key components such as the backend server, simulated terminal, and real-time feedback mechanism ensures a comprehensive educational tool that caters to users at all proficiency levels.

By using a modular architecture and adhering to best practices in design and development, the platform aims to provide a smooth and effective learning experience. The SRS document serves as a foundation for the development process, ensuring that all requirements are clearly defined and well understood. This will guide the creation of a reliable, user-friendly, and impactful learning tool that supports Linux command mastery in an intuitive and enjoyable manner.

References

1. Firebase Documentation (n.d.) https://firebase.google.com/docs
2. HTTP Methods for Communication (n.d.) https://www.javatpoint.com/http-methods
3. Node.js Documentation (n.d.) https://nodejs.org/en/docs/
4. Component Diagram (n.d.) https://www.geeksforgeeks.org/component-based-diagram/
5. UML Diagram - PlantUML (n.d.) https://plantuml.com/
6. UML Diagram - Canva (n.d.) https://www.canva.com/en_in/
7. VandenBrink, R. (2021). *Linux for Networking Professionals: Securely configure and operate Linux network services for the enterprise*. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10163502
8. Ovadia, S. (2016). *Learn Linux in a Month of Lunches*. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10280154
9. Miller, S. A. (2022). *Linux Administration Best Practices: Practical solutions to approaching the design and management of Linux systems*. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10162383
10. Calcatinge, A., & Balog, J. (2021). *Mastering Linux Administration: A comprehensive guide to installing, configuring, and maintaining Linux systems in the modern data center*. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10163616
11. Persson, J.-B. (2021). *Linux System Programming Techniques: Become a proficient Linux system programmer using expert recipes and techniques*. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10163418