Securing Services Infrastructure

MUHAMMAD ABDULLAH KHAN

01-134221-041

FAHAD BIN FAIZ

01-134221-020

Supervisor: Dr. Arif Ur Rahman

Bachelor of Science in Computer Science

Department of Computer Science

Bahria University, Islamabad

December 2025

Abstract

Securing critical service infrastructure has grown essential as cyber threats, system tampering, and illegal alterations continue to escalate. The goal of this project is to create a system that uses OpenSUSE Server's AIDE (Advanced Intrusion Detection Environment) to monitor, validate, and identify unauthorized changes in order to protect essential operating system functions.

The system implements cryptographic integrity checks, scheduled verification, secure baseline construction, and alert-based reporting to ensure that critical files remain uncompromised. A dedicated admin panel provides visibility into scan results, system health status, and validation logs.

This project increases infrastructure security by ensuring early detection of file modification threats, configuration tampering, and integrity breaches. The suggested approach provides a practical foundation for intrusion detection, secure configuration management, and enterprise-level infrastructure security.

Acknowledgments

We would like to express our heartfelt gratitude to our supervisor, Dr. Arif Ur Rahman, for his constant guidance, motivation, and invaluable insights throughout the development of this project. His expertise greatly contributed to shaping the direction and quality of our work.

We extend our appreciation to the Department of Computer Science at Bahria University, Islamabad, for providing the resources and academic environment necessary for this research.

We are also thankful to the developers and open-source communities behind AIDE, OpenSUSE, and Linux security tools, whose contributions form the backbone of this project.

We would also like to acknowledge our friends and classmates for their support and constructive feedback during various stages of this work. Finally, we express deep gratitude to our families for their patience, encouragement, and unwavering support throughout our academic journey.

MUHAMMAD ABDULLAH KHAN AND FAHAD BIN FAIZ

Islamabad, Pakistan

December 2025

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AIDE	Advanced Intrusion Detection Environment
FIM	File Integrity Monitoring
IDS	Intrusion Detection System
OSSEC	Open Source Security
SCAP	Security Content Automation Protocol
CIS	Center for Internet Security
NIST	National Institute of Standards and Technology
ISO	International Organization for Standardization
SMTP	Simple Mail Transfer Protocol
GUI	Graphical User Interface
API	Application Programming Interface
DB	Database
SSH	Secure Shell
PDF	Portable Document Format
UML	Unified Modeling Language
CPU	Central Processing Unit

Chapter 1

Introduction

Security in modern computing setups matters more than ever. Organizations keep building up their digital systems, so keeping things safe turns into a big deal. Server systems deal with all sorts of private data these days. They execute major services and operations that must remain on watch and guard at every hour. Nevertheless, with the expansion of relationships they become connected, and cyber attacks become intelligent, it becomes difficult to ensure files, setups and program files remain untouched. The attackers do not remain only in vulnerable areas of a network now. They go after stuff right on the machines. To be around longer, they modify or replace files. They hide what they are doing. They destroy the faith in the entire system.
This chapter gives an overview of the project known as the ***"Securing Services Infrastructure"***. The project will enhance the security of the server using the tools that will automatically check file changes. It provides the background, highlights the key issues it addresses, outlines the mission, and defines the general scope of the project [1], [2].

1.1 Project Background

Almost all the digital configurations involve server infrastructures as the basis. They handle key such components as authentication services, databases, and application logic. The overall reliability and belief in such systems depends much on the continuity and integrity of their base files stay. Even minor unauthorized changes to binaries, configuration files or service scripts are capable of causing significant security issues. Such problems may consist of privilege escalation, data backdoors that just hang about [3].

Basic security measures like firewalls, intrusion detection system and antivirus programs typically attack network traffic or patterns of known threats. They work well against an abundance of types of attack. Nevertheless, they are likely to ignore silent unsanctioned movements to the right within the operating system file setup.

FIM comes in to fill that significant gap. It keeps comparison of current state of system files with a previous trusted baseline. This method detects the changes, deletions, or additions made without authorization at an early stage [4].

In this project, we designed an entire File Integrity Monitoring system. It runs on the main monitoring part Advanced Intrusion Detection Environment which is abbreviated as AIDE. We implemented it on an OpenSUSE server. The Django backend is applied to perform secure. AIDE is a scanning, results interpreting, and audit records processing program. The frontend is a React-based development which provide an easy-going administrative interface. They are able to initiate scans, view the results and give detailed reports to forensics. All these pieces are made up into a strong, secured and scalable framework. It assists with maintaining tracking and integrity of systems [5].

1.1.1 Problem Description

Modern computing systems are subjected to unending security concerns such as unauthorized file alteration which may interfere with the system integrity and stability. Such modifications can go undetected using traditional security measures in real time thus leaving one open to attacks that will manipulate files without instant detection. Attackers can modify important binaries, configuration files or authentication keys thus obtaining a persistence or preventing security tools without detection.

The current tools produce either too many false positives or manually read through large log files which are time consuming and tiring to the administrator. Moreover, most traditional configurations have integrity baselines in the same system under monitoring. In case such a system is attacked, the files monitored as well as their baselines are subject to manipulation and thus cannot be detected [6].

The issue that this project will solve, then, is that there is a lack of a lightweight, automated,

and auditable file integrity monitoring system capable of identifying file manipulations by unauthorized persons in real time, with minimal false positives, and easy to respond to by an administrator. Through the combination of AIDE and Django and React, this project offers a smart monitoring system that is capable of proactive monitoring, systematic alerting, and secure audit tracing to OpenSUSE based systems.

1.1.2 Project Objectives

The project will be designed to implement an automated, transparent, and reliable file integrity monitoring system that enhances the host-level security. The core objectives are:

- **Automated Integrity Verification:** Configure and deploy AIDE on OpenSUSE Server to monitor critical files and detect any unauthorized modifications through checksum comparison and metadata validation.
- **Backend Management and Logging:** Develop a Django backed-end that is securely executes AIDE scans, analyzes reports and stores well-formatted scan results in a tamper-resistant database to be accessed later.
- **Frontend Visualization and Workflow:** Build a React-based dashboard of scan results, manual checks, file changes, generate signed PDF reports, and so on for administrators to view.
- **Alerting and Auditability:** Support real-time alerts and be completely alert trace of all approved and rejected modifications to be accountable and incident investigation.

By means of these objectives, the project can fill the gap between the conceptual file monitoring facilities and a practical and user-friendly security operations by providing a complete and interactive monitoring system [4].

1.1.3 Project Scope

The scope of this project is clearly stated to make it easy to manage and also align with the proposed objectives. The system is geared towards fundamental functionality and not excessive feature additions. Significant scope limits and characteristics are:

- **Platform Limitation:** The system will be built and deployed out of OpenSUSE Server

(Leap version) as per the departmental requirements. Cross-platform compatibility and packaging are out of the scope of the current project, even though the project is flexible to other Linux distributions.

- **Monitored Files:** Only critical files and directories such as `/etc/`, SSH keys, and important configuration files are monitored. Scanning of the entire system-wide is also not provided to ensure efficiency of performance and unnecessary overheads are minimized.
- **Backend Database:** The Django backend uses its own integrated database for storing scan results, detected changes, and reports. The system of this implementation does not have a concept of a remote database server and makes the system lightweight and locally self-contained.
- **Detection Focus:** The project emphasizes accurate detection and secure logging of unauthorized changes rather than performing automated recovery or file restoration. The repair and rollback processes will be counted as out of scope.
- **Deliverables:** The project deliverables consist of a functional web-based File Integrity Monitoring (FIM) tool comprising of AIDE and Django/React, a detailed documentation, architecture diagrams, and assessment outcomes.
- **Email Alerts:** The system provides automated email notifications to the administrator when scheduled scans are triggered. As an example, when an administrator sets an automated scan at 12 PM a day, a confirmation mail will be received as soon as the scan has commenced or completes. These email notifications make sure that they are known without the need of in-app alerts, making the dashboard lean and mean. The system sends email notifications to the administrator when a scheduled scan is done. As an example, when an admin puts an automated scan at 12 PM daily, the confirmation email will be sent when a scan beginns or completes. These email notifications make sure that there is awareness without using in-application notifications to keep the dashboard interface clean and focused.
- **No Custom Rule Creation:** Administrators are not allowed to define or modify custom AIDE rule sets for configuration files. The system preconfigures the rules of monitored paths and attributes and makes these rules consistent so that the risks of misconfiguration

are minimized, and the standardization between deployments is ensured.

- **Accept-All Change Policy:** The "Accept Changes" option enables administrators to accept all the changes that are identified simultaneously after viewing a scan report. This version does not support individual or selective acceptance of particular file changes. This design choice eases the process of updating the databases, and minimizes the risk of biased and inconsistent changes of the baselines.
- **Manual and Scheduled Scanning:** The system is compatible with manual and scheduled scans. The web interface can be used to schedule scans, and a manual scan enables the administrators review the integrity of the system in real time when required.
- **Report Generation and Logging:** A signed PDF report will be generated after every scan is automatically created, a summary of identified file modifications, and administrator actions. The reports are all stored in Django database and are accessible in the future on during audit or investigation.

The project has a realistic and narrow range with these limitations and functions scope and the end system is technically, fit, efficient and in accordance to the academic and operational objectives.

Chapter 2

Literature Review

File Integrity Monitoring (FIM) system is a basic element of cybersecurity that ensures that there are no unauthorized modifications to important files and configurations. The server based environment permits the malicious users to enhance their privileges and install backdoors, and steal data by alteration of files without monitoring. System integrity and accountability reliant on detection mechanisms which are major defense mechanisms to detect changes that are not authorized [7].

Over the last decade, numerous open-source and commercial FIM tools have been developed that offer distinct product mixes of configuration flexibility and detection accuracy and performance overhead and usability. Linux operating system relies onAIDE (Advanced Intrusion Detection Environment) and Tripwire as their main FIM tools but OSSEC, Wazuh, and Samhain offer more via the combination of FIM and the intrusion detection and compliance checks [8].

In this chapter, a critical analysis of FIM tools is done through evaluation of their architectural design and functionality capabilities. Their effectiveness in dealing with the modern system integrity issues. The research assesses the strengths and weaknesses of each tool to the requirements of the *Securing Services Infrastructure* with the help of AIDE on OpenSUSE Server project and finds out the current gaps in research and implementation.

2.1 Analysis of Similar Existing Systems

2.1.1 AIDE

It is an open-source tool called AIDE that generates a database using regular expressions that define the monitored files and the particular attributes such as permissions and checksums and timestamps. The tool is independent of operating systems and contains low resource usage that is used in academia and professional setting [9].

- **Strengths:** Easy configuration, quick scanning, high cryptographic hash capability (SHA-256, SHA-512).
- **Limitations:** No native real-time monitoring or centralized dashboard, weak alerting capabilities.

2.1.2 Tripwire

Tripwire is one of the first file integrity monitoring systems that is also offered in community and enterprise editions. The advanced customization options allow the system to provide users with the possibility to make complex rules and control the policy.

- **Strengths:** Enterprise version has centralized management and detailed reporting and high flexibility of rules.
- **Limitations:** The system requires a lot of processing power and its complex installation process complicates the use and maintenance of it by the administrators.

2.1.3 OSSEC/Wazuh

OSSEC and its offshoot, Wazuh, offer a unified system that incorporates file integrity checks with log analysis and rootkit detection and intrusion response support. The system provides real-time notifications in the form of centralized management dashboard that acts as one control interface.

- **Strengths:** Multi-agent support, comprehensive monitoring and reporting for distributed systems.

- **Limitations:** Consumes more system resources and difficult to install than isolated FIM software such as AIDE.

2.1.4 Samhain

Three primary features are offered by Samhain and these are file integrity, file log auditing and stealth detection of Unix-like systems. The system functions by client-server model that upholds encrypted communication among the parts of the system.

- **Strengths:** The product has tamper resistance, central management and stealth detection features.
- **Limitations:** More complex configuration increases CPU use to support encrypted communications; few GUI features.

2.1.5 OpenSCAP

OpenSCAP is an assessment system that relies on file integrity check that assists organizations to attain compliance and identify security vulnerabilities.

- **Strengths:** The system has full compliance auditing of the CIS NIST and ISO standards although exhibiting good potential in terms of expansion and connectivity to enterprise systems.
- **Limitations:** Not so much FIM, focused on compliance, heavy dependencies and slow scans.

2.1.6 Falco

Falco, which has been created by Sysdig, is not a conventional static FIM, but instead a runtime security monitor. The system is based on the analysis of system calls to identify unauthorized activity that occurs within containers and cloud systems.

- **Strengths:** The system provides real-time behavioral detection and shows outstanding container visibility and produces strong alert mechanisms.
- **Limitations:** The system does not monitor static files and requires complex setup procedures for non-containerized environments.

2.2 Comparative Study

Table 2.1: Comparative Analysis of Related Systems

Ref	Title	Key Insight	Project-Relevant Gap
[1]	Tripwire File Integrity System	One of the earliest FIM tools providing rule-based detection and centralized logging.	Heavyweight configuration; limited adaptability to lightweight Linux distributions.
[2]	OSSEC/Wazuh Framework	Provides host-based intrusion detection combining FIM, log monitoring, and alerting.	Complex to deploy; unnecessary overhead for small academic or lab environments.
[3]	Samhain Intrusion Detection	Offers centralized log management and file integrity validation.	Focuses mainly on enterprise-level setups; lacks modern web-based dashboards.
[4]	Auditd Linux Subsystem	Native Linux auditing for file access and modification tracking.	Logs are verbose and require manual correlation; lacks high-level visualization or filtering.
[5]	AIDE (Advanced Intrusion Detection Environment)	Lightweight and configurable FIM solution using cryptographic checksums for file monitoring.	Lacks real-time alerting and web-based review mechanisms; integration with dashboards not provided.
[6]	Wazuh Cloud Agent	Provides multi-host integrity scanning with cloud log storage.	Requires external cloud infrastructure; unsuitable for offline or isolated academic labs.

2.3 Summary

The comparative analysis demonstrates that AIDE functions as the best choice for lightweight file integrity monitoring when applied in academic institutions and other controlled settings. The deployment of Wazuh and Tripwire between their complex setup procedures and performance overhead results in making them less suitable for particular applications. AIDE provides an ideal solution because it operates with basic functionality and open transparency while working effectively with OpenSUSE Server for the project requirements. The project needs to link AIDE with Django backend React frontend system to create better usability and automated workflow solutions [10].

Chapter 3

Requirement Specifications

This chapter describes in detail the set of requirements needed for designing and developing the **"Securing Services Infrastructure"** project. It forms the technical backbone of the system, detailing the functionalities that the software needs to provide and the standards on which its performance is based. The requirement specifications form a bridge between the problem definition and the design of the system, mapping out all objectives, limitations, and expectations clearly prior to implementation.

System files integrity is an important security requirement in the present day computing environment. Any tiny unauthorized modification to a binary, configuration file or log can invalidate credentials, conceal backdoors or enable attackers to remain long-term. Some tools such as AIDE and Tripwire may have strong detection abilities, but they may not have the automation, centralized visibility, and usability components that may be required in practice. As a way of closing these gaps, this paper introduces an AIDE on OpenSUSE Server, coupled with a reactive frontend developed in React and a secure backend based on Django.

This chapter talks about the systems that will be analyzed, the limitations they are going to have and how the design is going to be enhanced in our proposed system. It does have both functional and non-functional requirements, which explain what is being done by the system as well as how efficiently and safely the system should be doing these functions respectively [8].

3.1 Existing System

There are many established solutions for FIM in today's landscape that can give partial coverage to system security. Most of them suffer from challenges that directly relate to usability, scalability, and integration. The following systems were studied and compared to identify the strengths and limitations.

3.1.1 System 1: Tripwire

Strengths:

- Mature and trusted FIM solution with comprehensive policy-based monitoring.
- Comprehensive reporting and detection accuracy.

Limitations:

- Complex to configure for academic or small-scale environments.
- No native web interface or alert management system.

3.1.2 System 2: OSSEC/Wazuh

Strengths:

- Provides log analysis, FIM, and intrusion detection in one platform.
- Provides dashboard integration and centralized alerting.

Limitations:

- High setup complexity and large resource footprint.
- Overwhelming alerts that cause operator fatigue.

3.1.3 System 3: Samhain IDS

Strengths:

- Cryptographic verification of file integrity and remote logging capabilities.

Limitations:

- Command-line only interface and outdated configuration methods.

3.1.4 System 4: Auditd

Strengths:

- Built-in Linux subsystem for tracking file access and modification events.

Limitations:

- Produces massive unfiltered logs with no visual representation and context.

3.1.5 System 5: AIDE

Strengths:

- Simple, fast and simple to configure file integrity checker.
- Strong validation is done using cryptographic checksums (SHA256, SHA512).

Limitations:

- No interactive management alerting system and GUI.

3.2 Proposed System

The solution proposed is a combination of AIDE reliability and the modern web technologies to enhance the visibility, usability, and automation. It presents an interactive dashboard, secured real-time communication of the back-end, and an automated schedule management option. The administrators are able to monitor scan histories, create monitored directories and even receive alerts through email in case of any anomalies [5].

3.2.1 The Proposed System Overcomes Existing System Limitations

- Brings out a convenient web dashboard on real-time scan management.
- Offers both manual and automatic scheduling of scans.
- Keeps historical logs of all scans for easy review.
- Ensures secure storage using a Django database.
- Sends email notifications to administrators without login alerts.

3.3 Comparison of FIM Techniques

Table 3.1: Comparison of File Integrity Monitoring Tools

Ref	Title	Key Insight	Project-Relevant Gap
[1]	Tripwire File Integrity System	Strong checksum validation and rule-based control.	Complex configuration; lacks web interface.
[2]	OSSEC/Wazuh IDS	Multi-agent host monitoring and alerting.	Heavy resource usage and alert fatigue.
[3]	Samhain IDS	Secure file integrity checks with cryptographic hashes.	Command-line only; minimal usability.
[4]	Auditd Linux Subsystem	Monitors system call-level file activity.	Raw, unstructured logs; no real-time visualization.
[5]	AIDE	Lightweight checksum verification engine.	No alerting or GUI; manual operation required.

3.4 Requirement Specifications

This system, "Securing Services Infrastructure," is intended to detect unauthorized file modifications, assure the integrity of server configurations, and maintain audit-ready reports. The requirements represent both what the system shall do and the operational constraints under which it shall do so.

3.5 Functional Requirements

3.5.1 FR-01: Admin Login

The solution shall use the Django authentication module to log in as administrator. Unauthorized users cannot access scan report and system settings.

3.5.2 FR-02: Manual File Integrity Scanning

The AIDE scans shall be manually triggered by the administrator using the dashboard. All scans will be processed, parsed and displayed on an interface.

3.5.3 FR-03: Scheduled File Scanning

The system shall support automated scanning at intervals set by the administrator (e.g. every 12 hours). The admin will receive an email alert automatically, but no in-app notification will be sent.

3.5.4 FR-04: Report Generation

A PDF report shall be automatically generated after each scan that summarizes the changes detected and the actions that can be taken.

3.5.5 FR-05: Manual Scan History

The system shall have a “History” tab through where admin can review any previously executed scan (manual) along with the timestamps and results.

3.5.6 FR-06: Scheduled Scan History

The scheduled scans shall be automatically logged in the Alerts tab for easy comparison between previous and recent scans.

3.5.7 FR-07: Add New Directories to Monitor

Administrators shall be able to extend monitoring coverage through the web interface by adding new directories or files in the AIDE configuration file.

3.5.8 FR-08: Accept Change Workflow

Administrators shall be able to accept all legitimate changes that the system detects. This acceptance automatically updates the AIDE baseline. It reflects the trusted configurations in the process.

3.6 Non-Functional Requirements

3.6.1 Performance

Scans for selected directories shall take a few minutes only. The parsing of the report and updates shall be shown on the web dashboard in 5–10 seconds.

3.6.2 Reliability

The system shall operate continuously, maintaining data integrity even in temporary network outages between the backend and database.

3.6.3 Security

All communications between the various components shall happen over encrypted connections. Such things as HTTPS or SSH do that. Access control to the database records will be kept and the records also remain audit-logged.

3.6.4 Usability

The frontend shall come with an intuitive design overall. It includes a clean layout for the dashboard. Then there is the history tab. Administrators can use it to review past scans pretty easily. They look over reports and any changes that got accepted.

3.7 Use Case Diagram

Use Case Diagrams depict the admin executing major AIDE monitoring activities: running scans and reports, managing AIDE directories, configuring systemd schedules, and handling admin account settings. The diagram illustrates that the system itself will automatically run scheduled scans via systemd timers, update the AIDE database on the Linux server, and send notifications via the email service after each scan or relevant event.

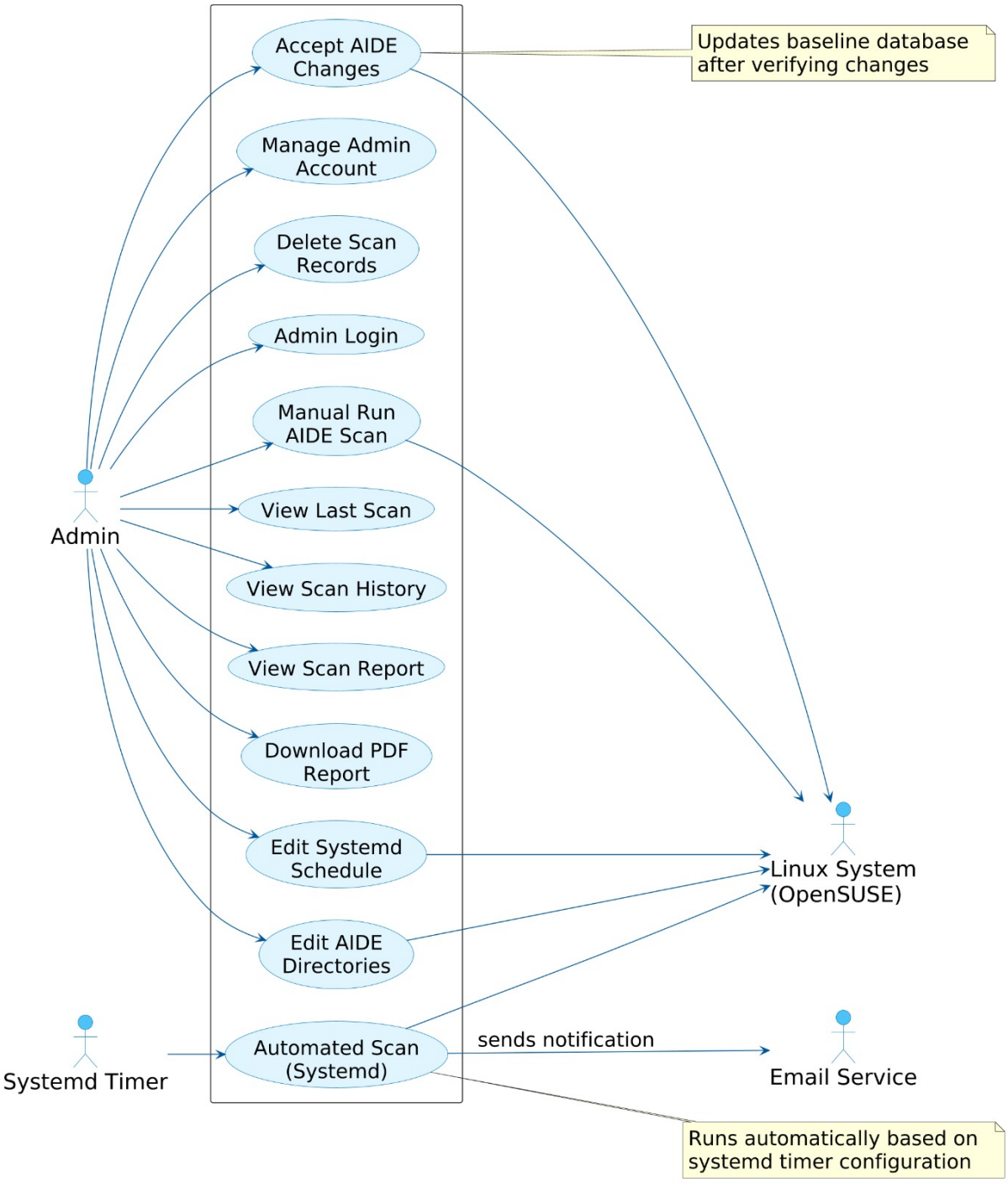

Figure 3.1: Use Case Diagram of the System

Table 3.2: Use Case 1: Admin Login

Use Case ID:	**UC-1**
Use Case Name:	Admin Login
Actor(s):	Administrator
Pre-Conditions:	Backend and database are running. Admin has valid credentials and network access to the web UI.
Priority:	High
Basic Flow:	1. Admin opens web UI. 2. Enters email and password. 3. Clicks Login. 4. Backend validates credentials and returns session token and after one hour it will automatically logged out . 5. Admin is redirected to Homepage.
Actor Actions	**System Response**
Admin submits credentials.	System authenticates and redirects to the Homepage.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin enters wrong password.	System shows error message and increments failed-login.
Admin requests password reset.	System sends a password reset code to the registered email.

Table 3.3: Use Case 2: Run Manual File Integrity Scan (RunCheck)

Use Case ID:	**UC-2**
Use Case Name:	Run Manual File Integrity Scan (RunCheck)
Actor(s):	Administrator
Pre-Conditions:	OpenSUSE server reachable, AIDE installed.
Priority:	High
Basic Flow:	1. Admin clicks `RunCheck` in the UI. 2. Django backend initiates AIDE runcheck command on CLI. 3. AIDE produces diff output. 4. Backend parses output, stores structured data in DB. 5. UI displays data.
Actor Actions	**System Response**
Admin clicks RunCheck.	System invokes AIDE, parses results, and updates dashboard with severity tags.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin starts scan while backend offline.	System returns error: "Connection error"; an event is logged for investigation.

Table 3.4: Use Case 3: Automated Scheduled Scans with Email Alerts

Use Case ID:	**UC-3**
Use Case Name:	Automated Scheduled Scans with Email Alerts
Actor(s):	Administrator, System
Pre-Conditions:	• Admin is authenticated and has access to system settings. • AIDE and Django backend are properly configured. • SMTP configuration and a valid admin email address are set.
Priority:	High
Basic Flow:	1. Admin navigates to the *Scheduling* section of the dashboard. 2. Admin sets a preferred time (e.g., daily at 12:00 PM) for automated scans. 3. The system saves this schedule using cron or systemd timers. 4. At the scheduled time, the backend triggers an AIDE scan automatically. 5. The system composes an email alert containing: • Scan summary (timestamp, total files checked, and changes detected). 6. The system sends the email alert to the admin's registered address. 7. Delivery status is logged in the audit trail.
Actor Actions	**System Response**
Admin sets time for automated scans.	System validates input and schedules the scan.
Scan executes at scheduled time.	System runs AIDE, parses output, and stores results.
Admin checks email after scan completion.	System sends summary email alert with report link.
System completes email delivery.	Delivery status recorded in the audit trail.
Alternative Course of Action	
Actor Action	**System Response**
Admin enters an invalid email address during configuration.	System displays an error message and prompts admin to enter a valid email address before scheduling can proceed.

Table 3.5: Use Case 4: View last Scan Summary

Use Case ID:	**UC-4**
Use Case Name:	View last Scan Summary
Actor(s):	Administrator
Pre-Conditions:	At least one scan record exists in the database.
Priority:	High
Basic Flow:	1. User opens dashboard summary. 2. React requests summarized results from backend. 3. Backend replies with last scan summary. 4. UI renders summary and links to details.
Actor Actions	**System Response**
User requests summary.	System returns summarized JSON and UI shows severity badges.
Alternative Course of Action (if any)	
Actor Action	**System Response**
No recent scan found.	System displays “No scans available” and suggests scheduling.

Table 3.6: Use Case 5: View History

Use Case ID:	**UC-5**
Use Case Name:	View Historical Change Log
Actor(s):	Administrator
Pre-Conditions:	Historical records retained in DB.
Priority:	Medium
Basic Flow:	1. User clicks History tab. 2. Backend returns paginated history. 3. UI displays list and allows filtering.
Actor Actions	**System Response**
User filters history by file or date.	System queries DB and returns matching entries.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Large result set requested.	System prompts to narrow query or requests confirmation to export.

Table 3.7: Use Case 6: Scheduled Scan History

Use Case ID:	**UC-6**
Use Case Name:	View Scheduled Scan History
Actor(s):	Administrator
Pre-Conditions:	- Admin is authenticated and logged in. - Automated scheduled scans are executed and stored in DB.
Priority:	High
Basic Flow:	1. Admin navigates to the Alerts/History tab. 2. System fetches all scheduled scan records from DB. 3. UI displays list of previous automated scans. 4. Admin may click "View Report" to open complete scan details. 5. Admin may click "Download Report" to download scan report as PDF.
Actor Actions	**System Response**
Admin clicks "View Report" for a specific scan.	System loads and displays the full scan report.
Admin clicks "Download Report" for a scan.	System generates a PDF report and initiates download.
Admin clicks delete icon for a scan.	System removes the selected scan entry from the DB.
Admin clicks "Clear History".	System deletes all previous scan records after confirmation.

Table 3.8: Use Case 7: View Report of Completed Scan in the UI

Use Case ID:	**UC-7**
Use Case Name:	View Report of Completed Scan in the UI
Actor(s):	Administrator
Pre-Conditions:	• Admin is logged into the system through the dashboard. • At least one scan (manual or automated) has been completed successfully. • Scan results have been parsed and stored in the database.
Priority:	High
Basic Flow:	1. Admin logs into the system and navigates to the *Scan History* section of the dashboard. 2. The system displays a list of completed scans with details such as date, time, and status. 3. Admin selects a specific scan entry to view detailed results. 4. System retrieves the scan report data from the database. 5. The detailed report is displayed in a structured format. 6. Admin may optionally export the report as a PDF using the "Generate Report" button. 7. The system confirms successful export and provides a download link.
Actor Actions	**System Response**
Admin navigates to *Scan History*.	System displays all previous scan entries from the database.
Admin selects a specific scan entry.	System retrieves and loads the detailed scan report.
Admin reviews report details.	System displays categorized file changes with timestamps and severity.
Admin clicks *Generate PDF Report*.	System generates and provides a downloadable PDF copy of the report.
Alternative Course of Action	
Actor Action	**System Response**
Admin selects a scan entry with incomplete or missing data.	System displays a "Report Unavailable" message and suggests re-running the scan.

Table 3.9: Use Case 8: Generate Audit PDF for Forensics

Use Case ID:	**UC-8**
Use Case Name:	Generate Audit PDF for Forensics
Actor(s):	Administrator, Auditor
Pre-Conditions:	Scan data present and backend PDF engine available.
Priority:	Medium
Basic Flow:	1. Admin requests PDF for a scan or period. 2. Backend compiles signed PDF including metadata and explanations. 3. PDF ready for download.
Actor Actions	**System Response**
Admin clicks Generate PDF.	System builds deterministic PDF and returns file or error.
Alternative Course of Action (if any)	
Actor Action	**System Response**
PDF generation fails.	System logs error and suggests retry or smaller report scope.

Table 3.10: Use Case 9: Accept all Changes

Use Case ID:	**UC-9**
Use Case Name:	Accept all Changes
Actor(s):	Administrator
Pre-Conditions:	Admin has reviewed change details .
Priority:	High
Basic Flow:	1. Admin clicks `Accept Change` . 2. Backend updates the DB with the last scan . 3. System logs the acceptance event.
Actor Actions	**System Response**
Admin accepts change .	System updates baseline .
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin cancels acceptance.	System leaves baseline unchanged .

Table 3.11: Use Case 10: Manage Account

Use Case ID:	**UC-10**
Use Case Name:	Manage Account
Actor(s):	Administrator
Pre-Conditions:	Admin has superuser privileges.
Priority:	Medium
Basic Flow:	1. Admin navigates to user management. 2. Edit account details like password . 3. Changes are validated and saved in DB.
Actor Actions	**System Response**
Admin edit account.	System validates data and stores account with assigned role.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin tries to create duplicate username.	System rejects and prompts for a unique username.

Table 3.12: Use Case 11: Configure Monitoring Rules

Use Case ID:	**UC-11**
Use Case Name:	Configure Monitoring Rules
Actor(s):	Administrator
Pre-Conditions:	Admin has access to directories in config file.
Priority:	Medium
Basic Flow:	1. Admin adds/removes file paths to monitor. 2. Select rules and attricutes to monitor . 3. Saves configuration; backend updates AIDE rules.
Actor Actions	**System Response**
Admin edits monitored file list.	System updates rules and schedules baseline update if requested.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin provides invalid path.	System validates and returns an error message explaining the expected format.

Table 3.13: Use Case 12: Admin Changes Automated Scheduled Scan Time

Use Case ID:	**UC-12**
Use Case Name:	Admin Changes Automated Scheduled Scan Time
Actor(s):	Administrator, System
Pre-Conditions:	• Admin is logged into the system with proper privileges. • An automated scan schedule is already configured in the system. • The backend scheduler (cron or systemd timer) is active and functional.
Priority:	Medium
Basic Flow:	1. Admin navigates to the *Scheduling* section of the dashboard. 2. Admin views the currently active automated scan schedule. 3. Admin selects the option to modify the scan time. 4. System prompts the admin to input a new preferred time (e.g., 2:00 PM daily). 5. Admin enters the new time and clicks *Update Schedule*. 6. System validates the entered time format and updates the schedule in the backend. 7. Backend reconfigures the cron/systemd timer to reflect the new schedule. 8. System confirms the successful update and displays a notification on the dashboard. 9. Updated schedule is recorded in the audit trail for future reference.
Actor Actions	**System Response**
Admin navigates to scheduling section.	System displays current scan schedule and modification options.
Admin enters a new scan time.	System validates format and updates cron/systemd configuration.
Admin confirms update.	System saves new time, reinitializes scheduler, and logs the change.
Alternative Course of Action	
Actor Action	**System Response**
Admin enters an invalid time format or a past time.	System displays an error message and prompts the admin to provide a valid future time in correct format.

Table 3.14: Use Case 13: Housekeeping — Delete Old Records

Use Case ID:	**UC-13**
Use Case Name:	Housekeeping — Delete Old Records
Actor(s):	Administrator
Pre-Conditions:	Old scan results available.
Priority:	Low
Basic Flow:	1. Admin selects delete icon in front of each scan. 2. Backend moves old records to archive or deletes per policy. 3. Operation logged.
Actor Actions	**System Response**
Admin initiates cleanup.	System executes housekeeping job and logs summary.
Alternative Course of Action (if any)	
Actor Action	**System Response**
Admin tries to delete without backup.	System warns and requires confirmation.

Chapter 4

System Design

System design plays a significant role in converting the conceptual needs of the project, which is the project, “Securing Services Infrastructure”, into a vivid roadmap that would be used during implementation. The purpose of this system is to make sure that files are not altered by anyone, this is why it is used to identify any unauthorized alterations on the files. The design is security oriented, reliable and easy to operate, this stage defines the interaction of the web-based administrator interface, the django backend, the AIDE file integrity engine, and automated scanning services to deliver. monitors and allows a professional environment to be easily managed.

The system architecture has 3 key layers: React frontend, Django REST API backend, and the OpenSUSE server services which are comprised of AIDE and SQLite database, SMTP, and Systemd timetables. All these layers are meant to deal with a particular responsibility. Frontend provides a safe interaction and presentation of scan outcome, the backend regulates identification, scan execution, alteration acceptance, creation of PDF reports and alerting, and the server layer has to be sure that trusted file integrity scanning, information storage, and automaton is executed. The modularity gives it simplicity of maintenance, enhanced scalability, and assists in making sure that system security is not compromised by vulnerabilities at the UI.

The ability of the design approach to focus on both manual and automated security enforcement is also a major strength of the design approach. As an example, the administrators can scan, accept changes and do system parameter management via the web interface. Simultaneously, automated periodical scanning has also been incorporated into the design, based on Systemd

timers, and alerts that are generated are assisted by email reporting using SMTP. The design has role-based authentication in place to make sure that there is high cybersecurity, safe hashing of credentials, encrypted HTTPS-based communication and strict timeliness of the session. The system architecture incorporates both an organized system design and a robust backend control robust system architecture in addition to easy to use administration portal to effectively secure important file assets, as it continues to provide smooth control over operational matters to the administrators.

4.1 System Architecture

The structure of the system architecture of the solution called the "Securing Services Infrastructure" is modular, a three-tier design, and also divides the concerns into specific layers: frontend interface, backend logic, and database persistence. This ensures that there is scalability, maintainability and high fault isolation among components.

The architecture integrates AIDE with a Django backend that safely scans integrity of an OpenSUSE Server. This backend communicates with the React-based frontend by means of RESTful APIs to present summarized results, offer the option to administrators to accept legitimate changes, and create audit reports.

The key components of the system are:

- **Frontend (React):** It presents a simple and user-friendly dashboard, on which the scans may be created, the results may be analyzed, and such activities as "Accept Change" or "View Report" may be taken.
- **Backend (Django):** This serves as the control centre which initiates AIDE scans system calls, interprets the resultant information and communicates with the database.
- **Database (MySQL):** A database is a secure, tamper resistant store of scan data (structured), audit logs, user information and configuration parameters.
- **AIDE on OpenSUSE Server:** Checks the actual file integrity by cryptographic hash and comparing metadata.

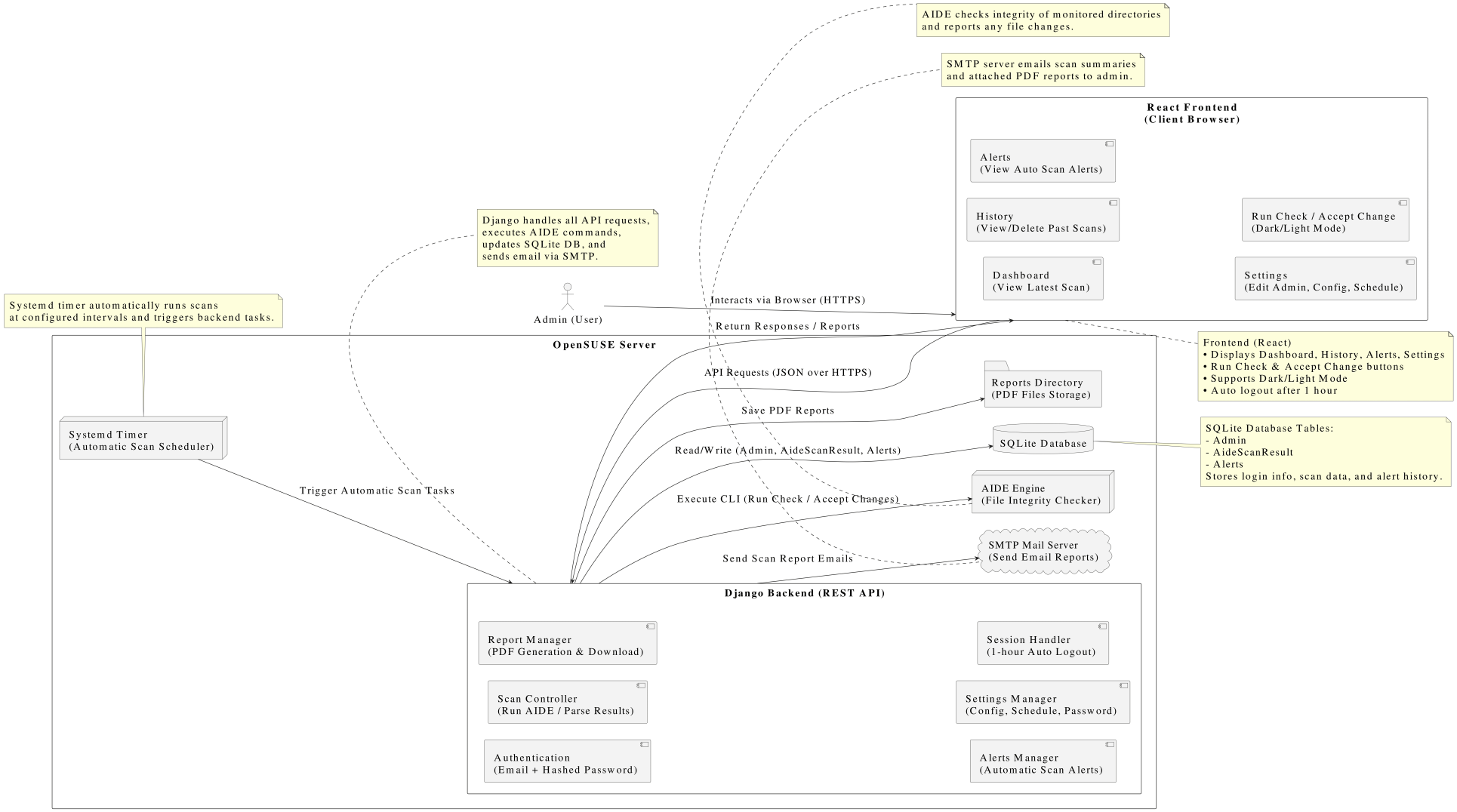

Figure 4.1: System Architecture Diagram

4.2 Design Constraints

The architectural design of this system should meet multiple operational, environmental, and security restrictions which affect architectural and implementation decisions. These limitations are to keep the end product viable and implemented in the field.

4.2.1 Cross-Platform Compatibility

The backend is also optimised with the OpenSUSE Linux and keeping in mind the portability that in future it may port to other Unix based systems such as Ubuntu, Fedora and Debian. Python and Django make it possible to be flexible with different platforms. AIDE configuration scripts might have to be slightly altered according to file system path and package management systems.

4.2.2 Security Constraints

One of the design principles was security. Communication among the system components is HTTPS or SSH tunnel-encrypted, i.e., between the frontend and the backend as well as the database. Authentication procedures are based on the principles of role-based access to the critical data; therefore, cylindrical credentialing is used to store credentials with the salted hashing.

4.2.3 Performance Constraints

Since integrity checking may be costly, system design restricts scans to high value directories like /etc, /usr/bin and path of SSH keys. The systemd scheduler handles frequency to avoid overloading a CPU. Database access and parsing functions are optimized to have the lowest possible response time between AIDE output and UI update.

4.2.4 Usability Constraints

Its web interface is intentionally sparse, focusing primarily on reading, ease of use and obvious visual feedback as to the extent of file modification. A maximum readability is ensured by

adhering to the accessibility standards, including the color contrast and the font size.

4.2.5 Scalability Constraints

The architecture, though designed to be deployed in academic and small enterprise, has the ability to scale horizontally, meaning that in the future the multiple servers being monitored can report to a central backend and a database instance with only slight adjustments.

4.3 Design Methodology

The design methodology that is used in this project is the Spiral Model. The Spiral Model is more appropriate to systems that demand a constant improvement, risk analysis and successive refinement, which is the case of a security-oriented application like a file integrity monitoring system.
The repeat cycles are the methodology and each cycle comprises the four major activities:

1. Planning

In this stage, objectives and needs that will direct every development stage are determined. This involves the definition of system characteristics, user behaviors, security requirements and general work requirements. Planning helps to determine the scale of every cycle and make development in small steps.

2. Risk Analysis

The implementation is evaluated based on the possible technical or security threats. These risks can be related to the performance of the system, misconfigurations, authentication related, or serverlevel execution. This will give a greater chance of making decisions between paths of design, which will be safer, and embrace alternative preventive actions.

3. Engineering

The real development of the system according to the planned requirements is this stage. It entails deployment of the frontend interfaces, the backend logic, database activities, and system

functionalities. With every iteration, a more complete and advanced version of the system is produced.

4. Evaluation

The developed features are tested and reviewed after every cycle. The accuracy, usability, and dependability of the system are checked. The next version intends to improve, depending on the feedback received at this stage.

Summary

The Spiral Model enables the project to expand in a gradual manner, and risks are managed in a consistent manner and the quality of the system is also improved. It is flexible by virtue of its iterative character which makes it an effective process in the process of creating security oriented monitoring solutions.

4.3.1 Workflow Diagram

This Workflow Diagram is an overview of the overall communication between an administrator and the system. It works via the login of the admin with verified credentials. After authentication the administrator is redirected to the dashboard, where the last scan result summary can be seen and links to other important features, including manually executing AIDE scans, accepting changes to the baseline database, settings, and scans history are. The scheduled scans are also automatically performed via systemd and the results are stored in the database with notifications sent out via email on detecting any changes. It also enables the administrator to read or download report of current scan, scan history management by deleting a particular report or deleting all scan history as well as set up system settings regarding AIDE directories, monitoring regulations and default scan schedules. Automatic logout of the system will be one hour after the admin logs in.

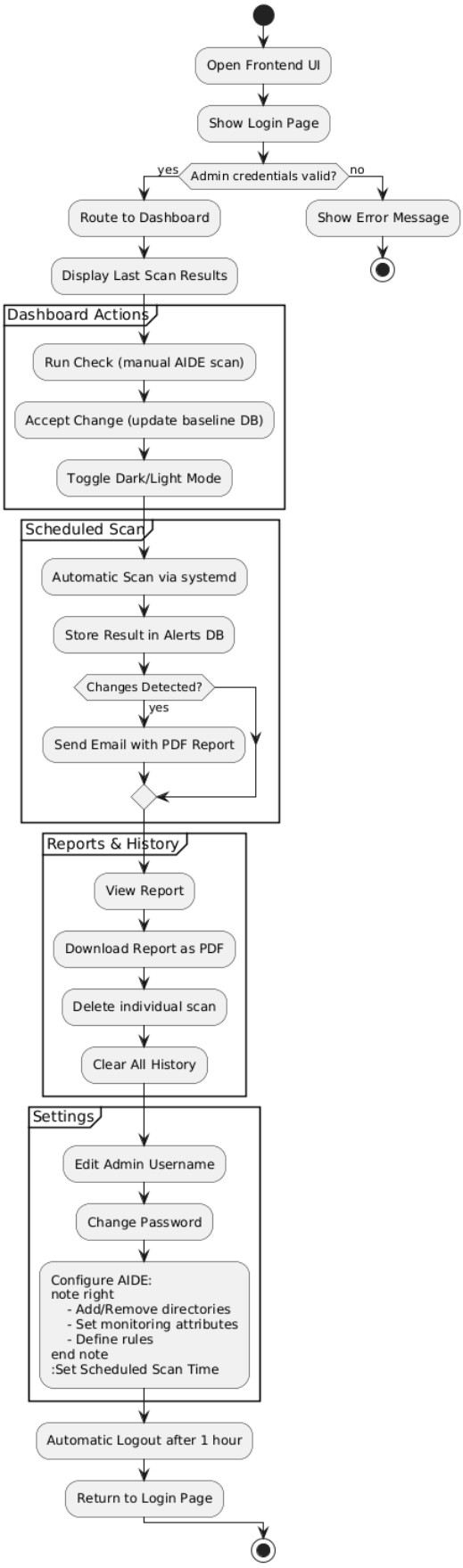

Figure 4.2: Workflow Diagram of the System

4.3.2 Activity Diagram

The Activity Diagram gives the general flow of the entire system. It emphasize interactions between the administrator and the automated parts. After successfully entering without any problems, the admin is redirected to the dashboard. There, it has other features which are independent such as Run Check, View Report, Accept Changes, Settings and History. They both work parallel to each other.

- **Run Check**: Manually performs an AIDE scan and updates the dashboard assuming that the result is valid, otherwise it will show an error.
- **View Report**: Opens the stored scan reports or a new PDF file generated.
- **Accept Change**: Substitutes the current state of the baseline database with the latest findings of the scan, and gives a notification of successful operation or an error message.
- **Settings**: Provides the possibility of changing AIDE settings, setting profile settings and the automatic scan timer based on Systemd.
- **History**: View history of all manual and automated scan records.
- **Automatic Scheduled Scans**: Systemd runs scans automatically at the time set up, and sends email notification if SMTP is enabled.
- **Auto-Logout Policy**: System auto-logout is exactly one hour to ensure security, regardless of user activities.

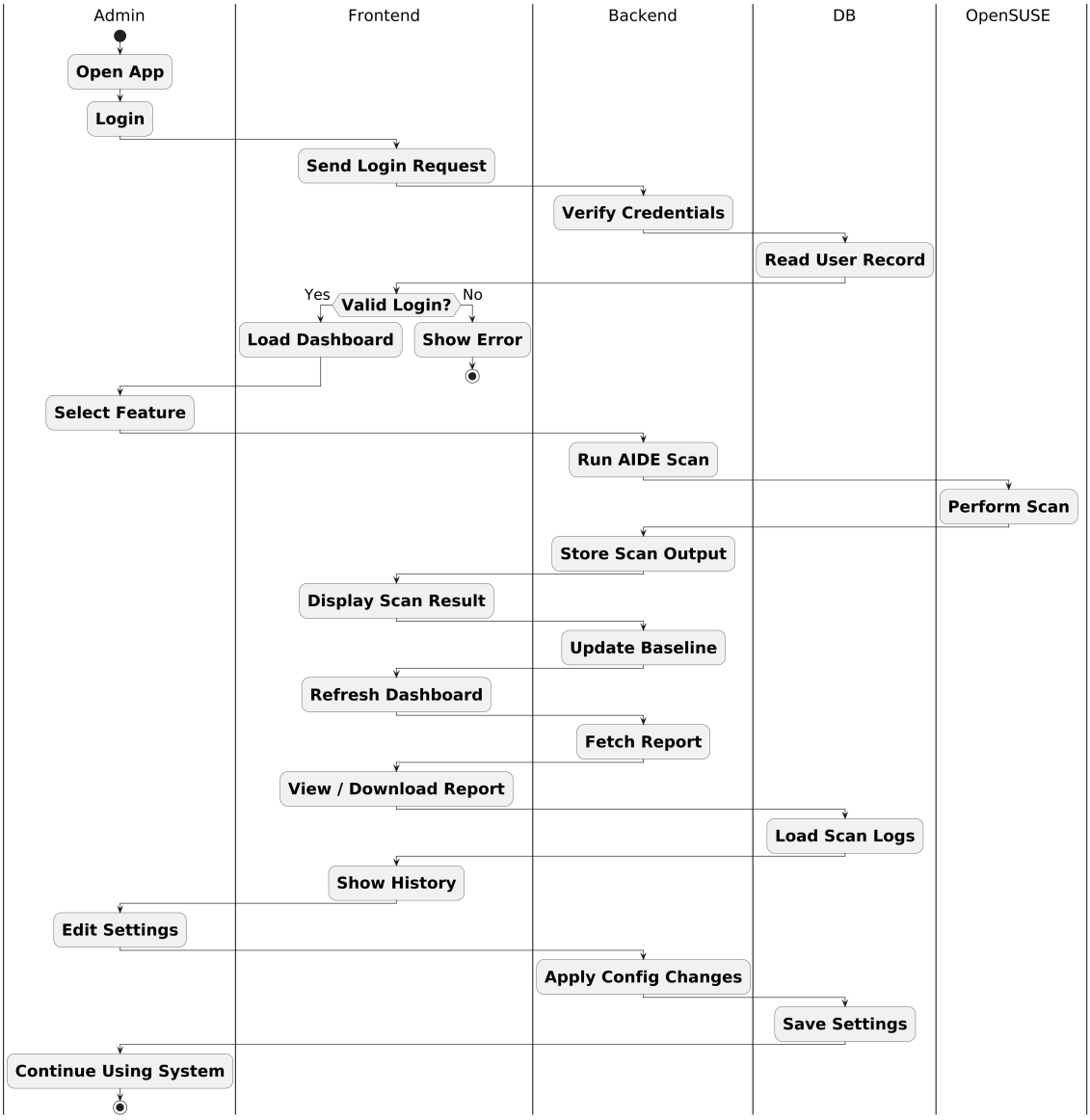

Figure 4.3: Activity Diagram of the System

4.4 Sequence Diagrams

4.4.1 Login Sequence Diagram

This diagram describes the workflow of an admin login. An admin inputs credentials; the Django backend checks these credentials, a session token is generated, and a session timeout in one hour is turned on. After an hour, the session automatically expires, and the admin gets logged out.

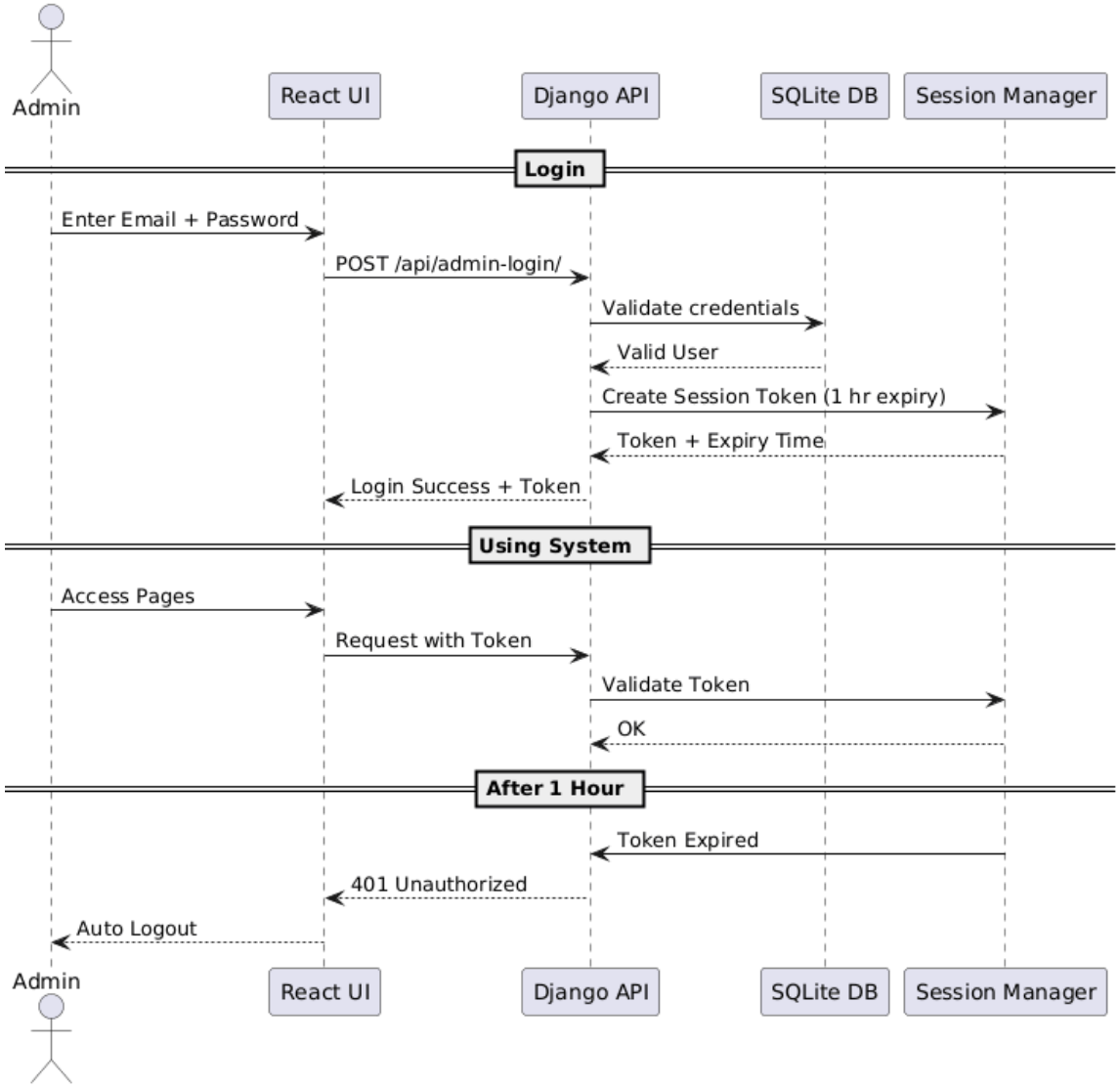

Figure 4.4: Login Sequence Diagram

4.4.2 Manual Scan (RunCheck) Sequence Diagram

This sequence diagram describes the process of an admin-initiated manual scan. The front end sends a run-scan request to the Django back-end, which runs AIDE, parses the results, does the

necessary database updates, and returns the scan results back to the front end.

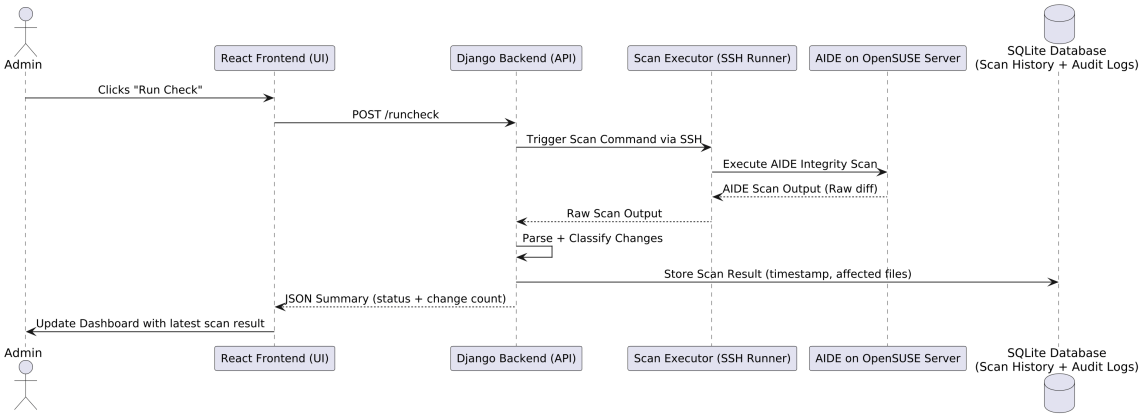

Figure 4.5: Manual Scan (RunCheck) Sequence Diagram

4.4.3 Automatic Scheduled Scan Sequence Diagram

The auto-scheduled scan is triggered by the Systemd Timer. Django invokes AIDE, summarizes results and stores those in the Alerts model, and optionally emails a report to the admin.

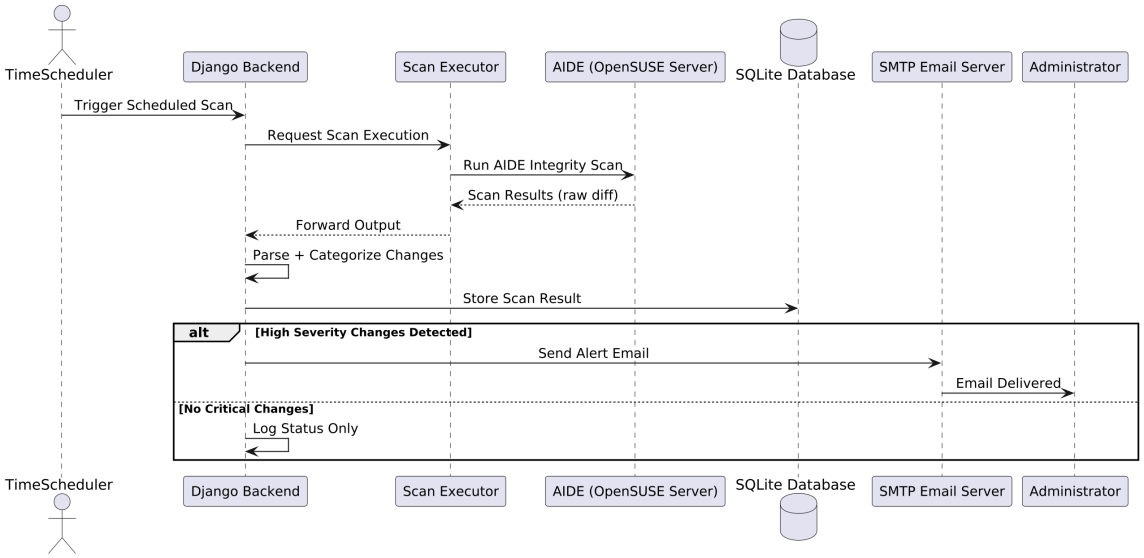

Figure 4.6: Automatic Scheduled Scan Sequence Diagram

4.4.4 History and Alerts Sequence Diagram

This diagram depicts how the frontend retrieves past scan results. History tab fetches manual scans from the AideScanResult model. Alerts tab retrieves auto-scan alerts from the Alerts model.

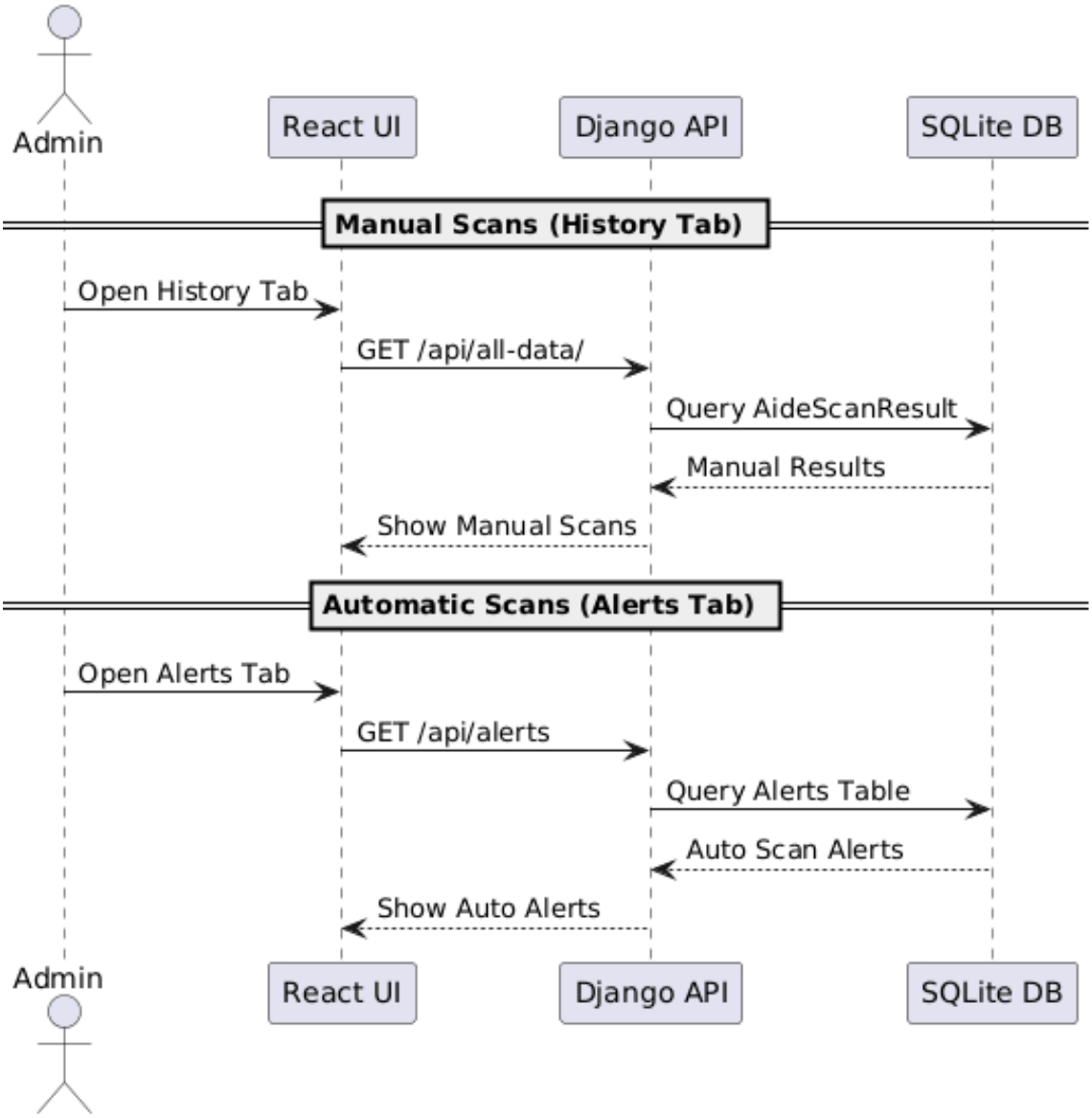

Figure 4.7: History and Alerts Sequence Diagram

4.4.5 View and Download Report Sequence Diagram

This diagram depicts how the admin views the detailed scan results or downloads a PDF report. The backend parses AIDE results, generates a PDF if needed, and returns it to the frontend.

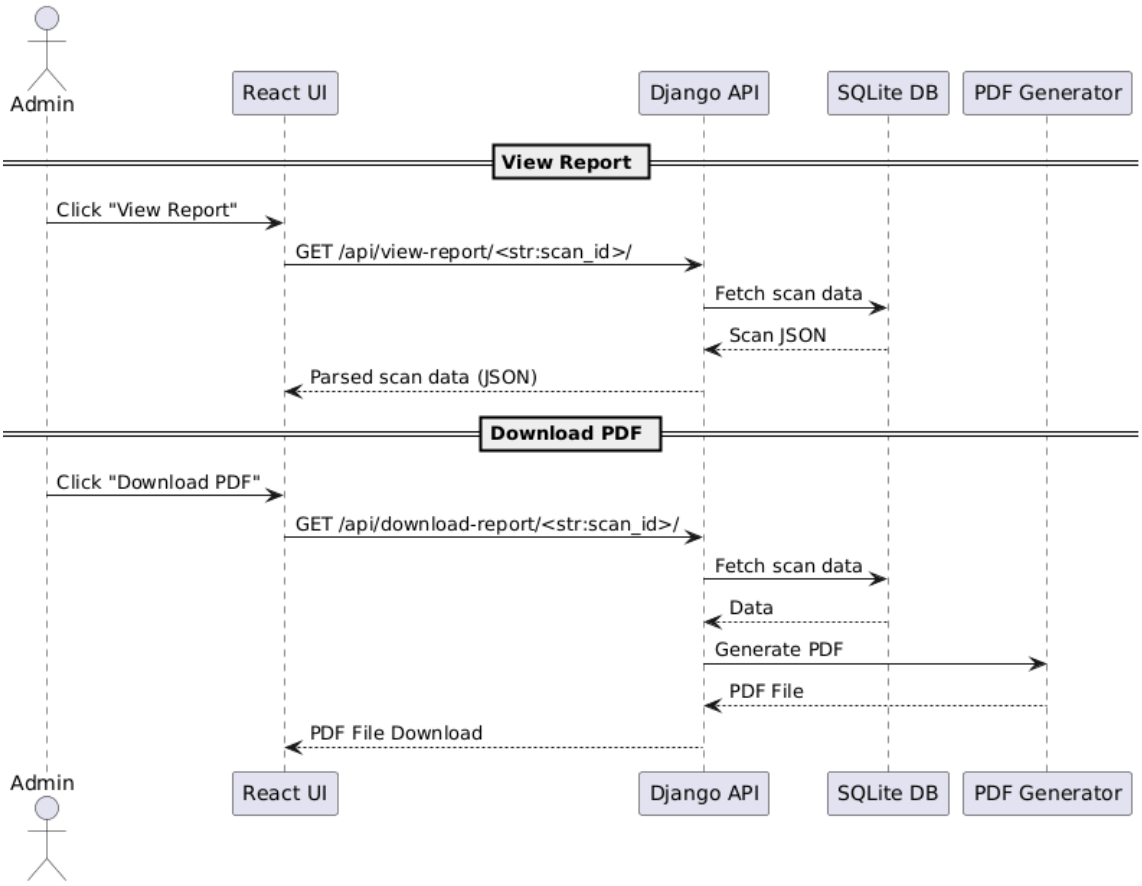

Figure 4.8: View and Download Report Sequence Diagram

4.4.6 Accept Changes Sequence Diagram

This diagram depicts how the admin accepts the changes of the last scan. Acceptance updates the AIDE baseline database and triggers a new scan right away to validate the updated state.

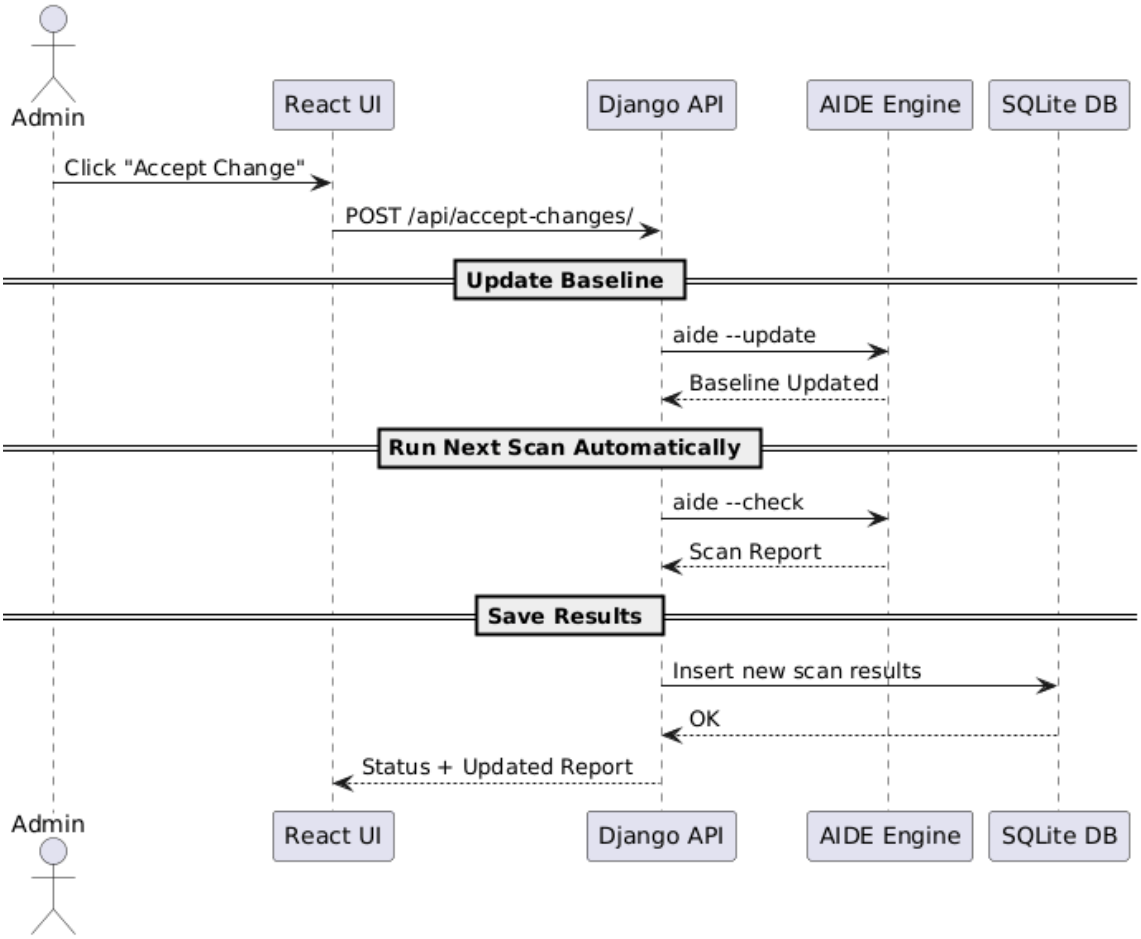

Figure 4.9: Accept Changes Sequence Diagram

4.4.7 Settings Sequence Diagram

This is a sequence diagram for editing the username, changing the password after verification, editing the AIDE configuration directories, and updating the schedule of automatic scanning by Systemd.

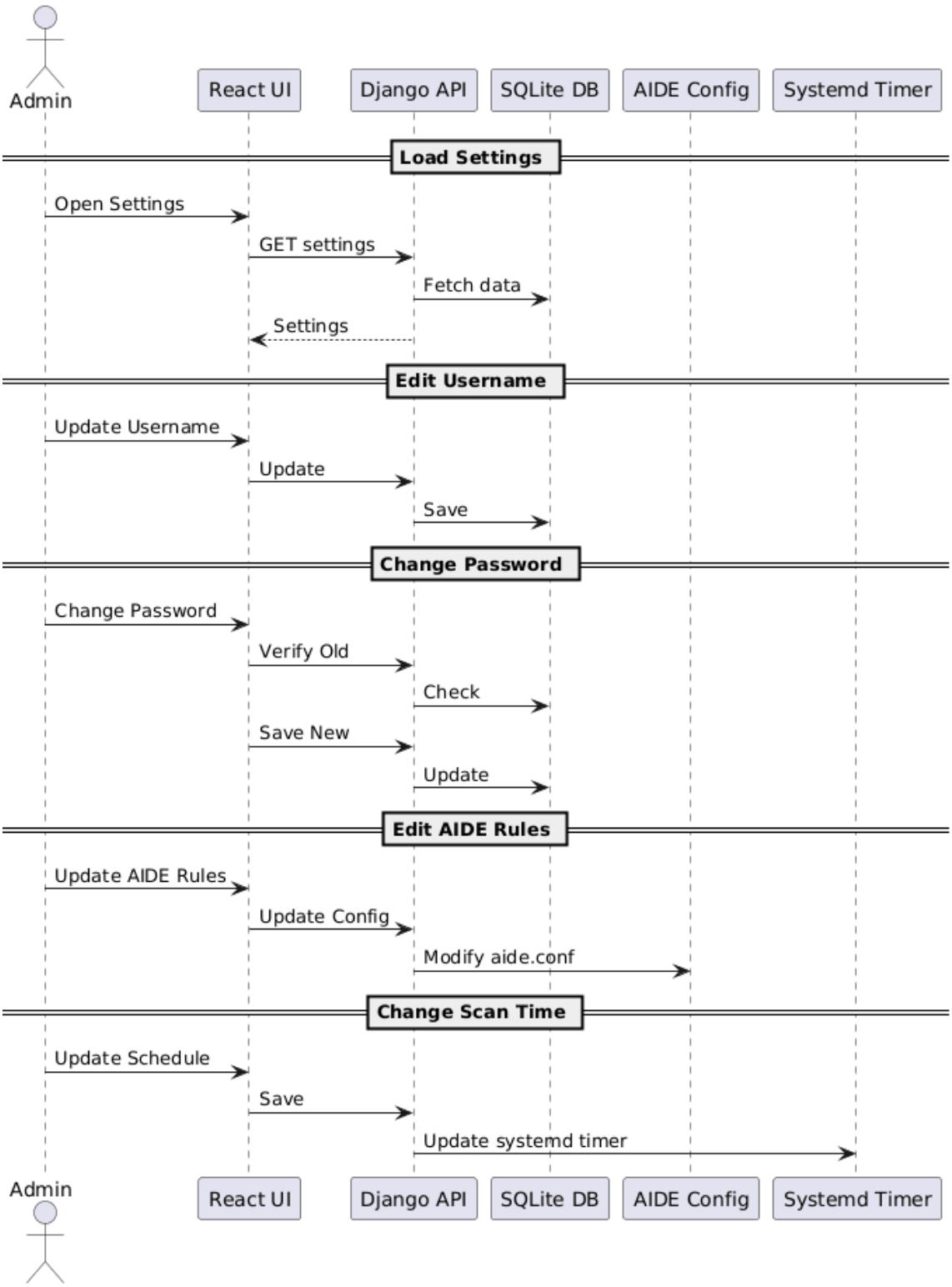

Figure 4.10: Settings Sequence Diagram

4.5 Low Level Design

The Low-Level Design of the AIDE-based FIM System provides a detailed view of the internal structure, including the interaction of components, back-end data models, and user interface functional mappings. This stage determines the implementation process and the manner in which every module handles requests and retains scan results and alerts to subsequent use.

The AIDE Scan Execution Module creates both manual and scheduled scans, authenticates the AIDE output and stores all the results in the system database. Using the Alert and Notification Module, high-severity file changes are processed and alert entries are constructed which alert administrators on the dashboard and via email with the use of SMTP. In the meantime, the Admin Authentication Module will be used to ensure that a secure access to the system by verifying the legitimacy of the administrator credentials and automatically logging out after one hour.

Frontend modules contain the Dashboard, History, Settings, and Alerts pages that are highly specialized controllers necessary to complete certain administrative tasks such as communicating with backend APIs to retrieve scan output, displaying past results, updating baseline, AIDE settings, and PDF reports. Some of the important backend structures that enable data integrity are the AideScanResult model that is used to store scan metrics and the CustomUser model that is used to authenticate and enforce privileges.

A combination of all of these modules forms a coherent and secure workflow wherein integrity scans are created, processed, documented, and ultimately provided to give system administrators a continuous control to act and react to unauthorized changes to. infrastructure files.

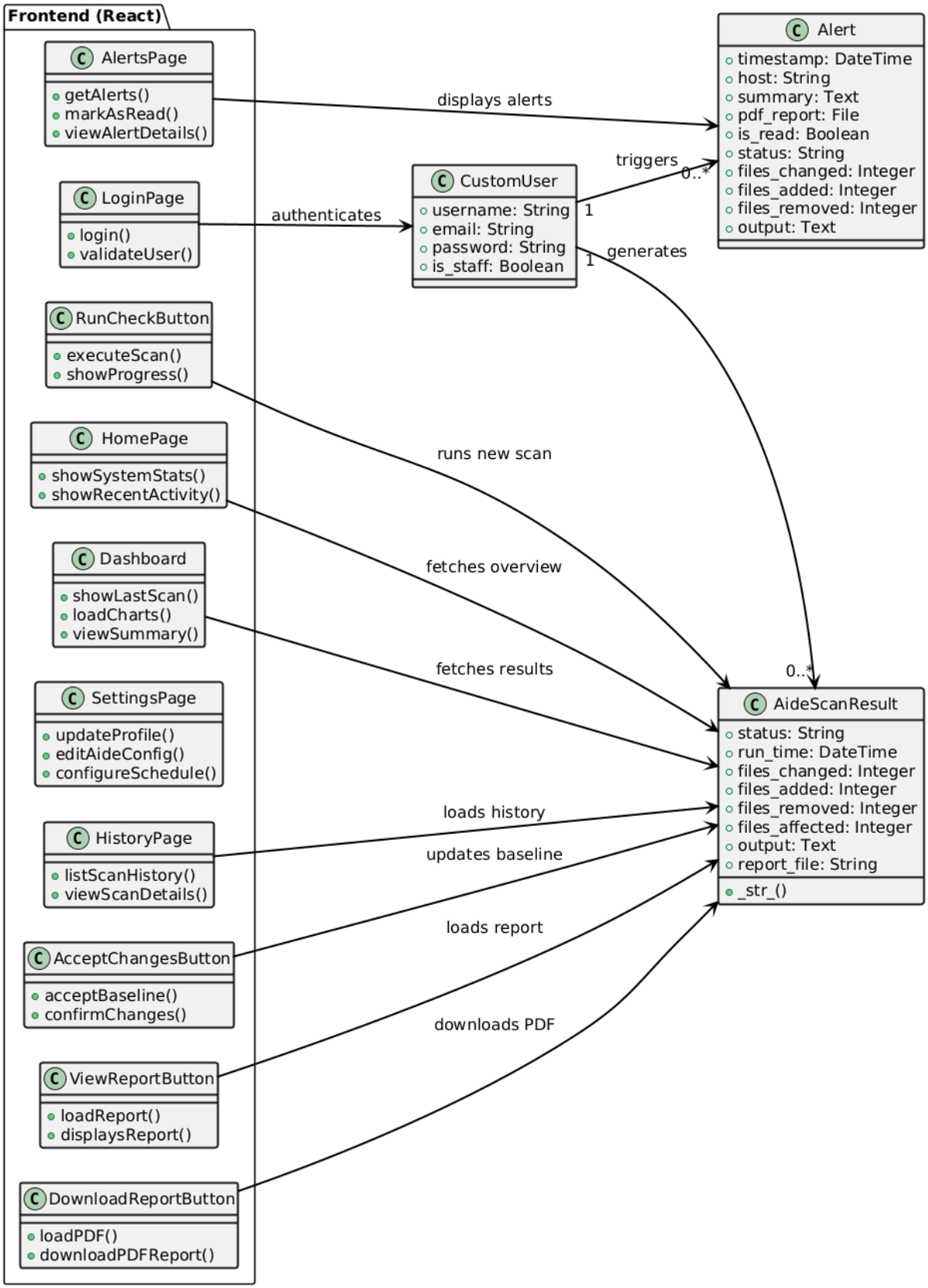

Figure 4.11: UML Class Diagram

4.6 GUI Design

This part is dedicated to the graphical user interface that a system administrator would interface with with ease with the file integrity monitoring system. It emphasizes simplicity, security and speed to the main features: manual scans, monitoring automation, access to reports, configuration modifications, and visibility of status.

4.6.1 Login Page

Only the administrators can gain access to the system in the login screen. This contains a design of very simple nature with limited authentication in terms of username and password. All attempts that are invalid come up with security validation messages that would safeguard the access to the system.

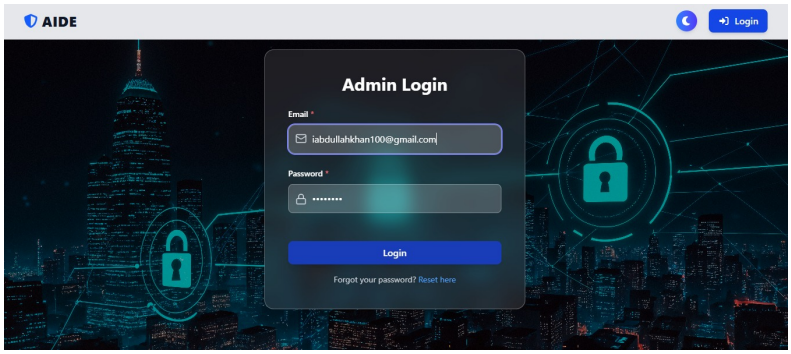

Figure 4.12: GUI – Admin Login Page

4.6.2 Home Screen – Overview

It is the interface that one gets after the login. It gives quick methods to open a new check manually, open the last reports and go to the dashboard or the settings. It provides efficiency in getting workflow going on the administrative side.

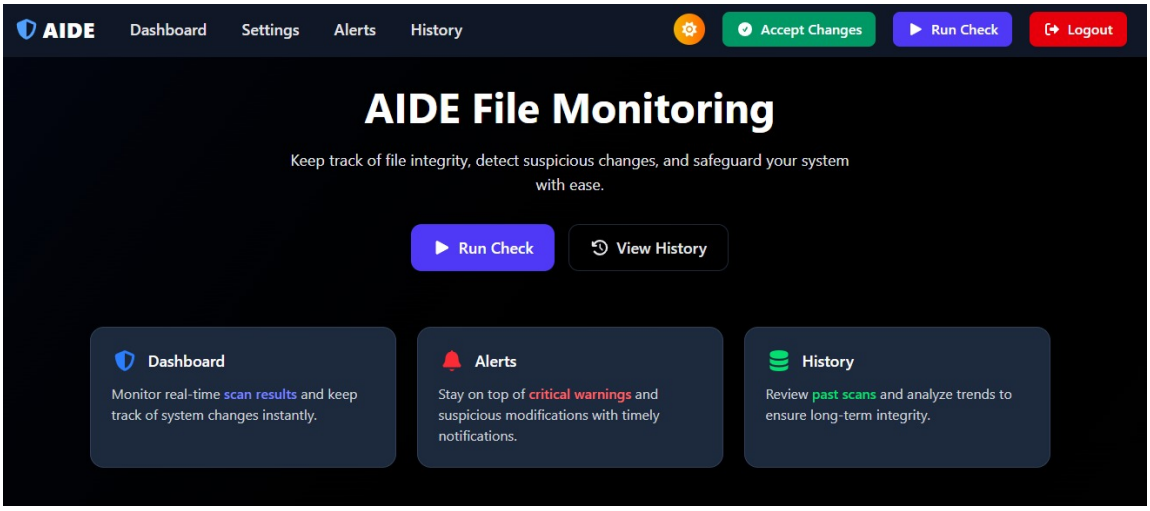

Figure 4.13: GUI – Home Screen (Overview and Primary Actions)

4.6.3 Home Screen – Status View

This view displays in real-time the system health indicators that are determined by the recent scanning of AIDE. It graphically indicates the state of the integrity of the system being intact or not and helps the admin to implement the rightful step in case it is necessary.

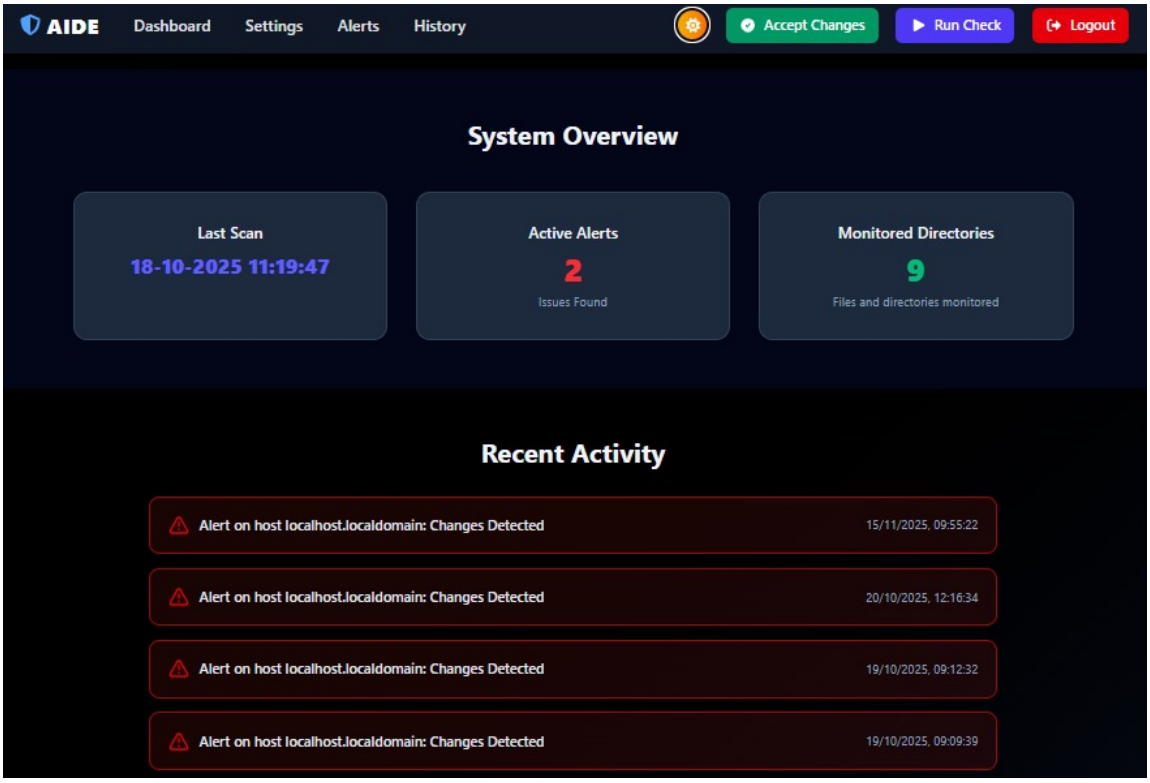

Figure 4.14: GUI – System Status Summary from Latest Scan

4.6.4 Admin Dashboard – Scan Summary

The dashboard gives a summary outcome of AIDE integrity scans. It categorizes the changes identified as files that are modified, added or removed so as to easily prioritize the threats by the administrators.

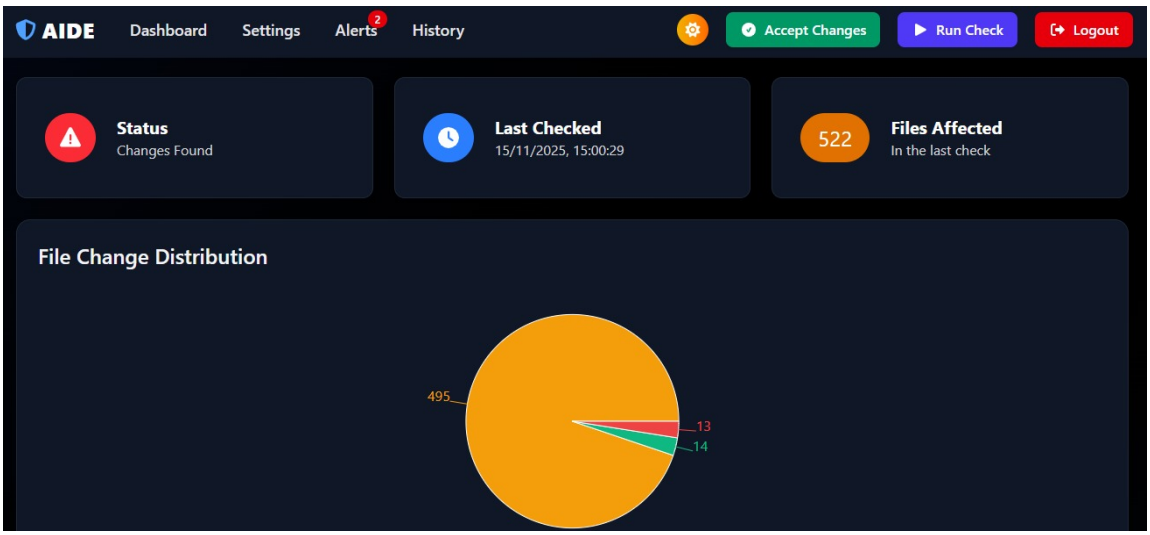

Figure 4.15: GUI – Dashboard Showing Categorized Scan Summary

4.6.5 Admin Dashboard – Detailed Results

This section allows deep inspection of each suspicious file. It provides file paths, timestamps, and types of changes which assist the admin in verification and further action.

Figure 4.16: GUI – Detailed Results with File-Level Integrity Logs

4.6.6 View Reports

The admin can preview or download the generated PDF reports that summarize every AIDE scan run. This will provide documentation and traceability for audits, investigations, and compliance requirements.

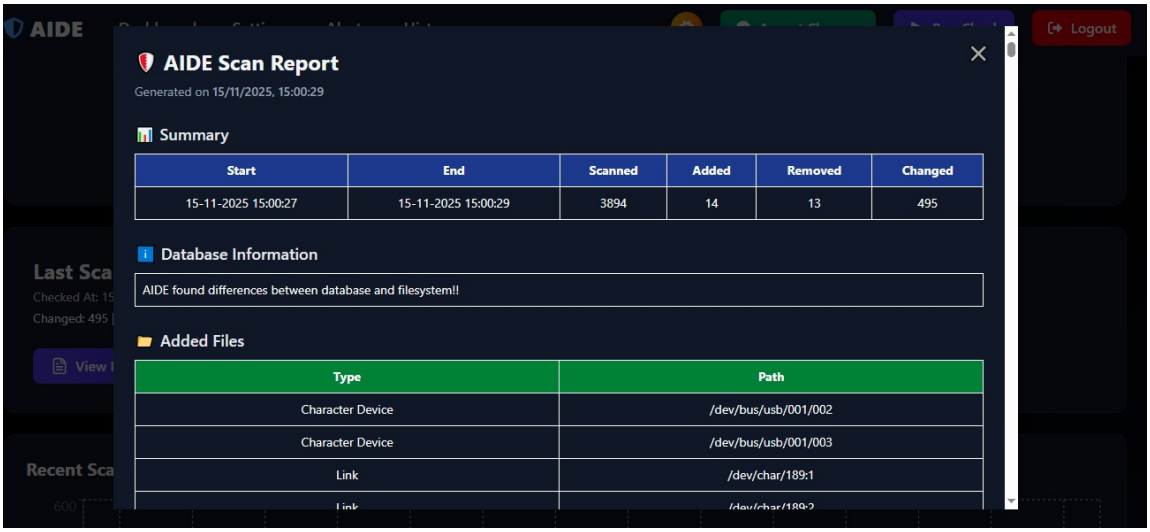

Figure 4.17: GUI – Report Management and PDF Access

4.6.7 Manual Scans History

This page enumerates timestamps and outcomes of all manual executed in the past. It helps to recognize recurring threats and operation patterns.

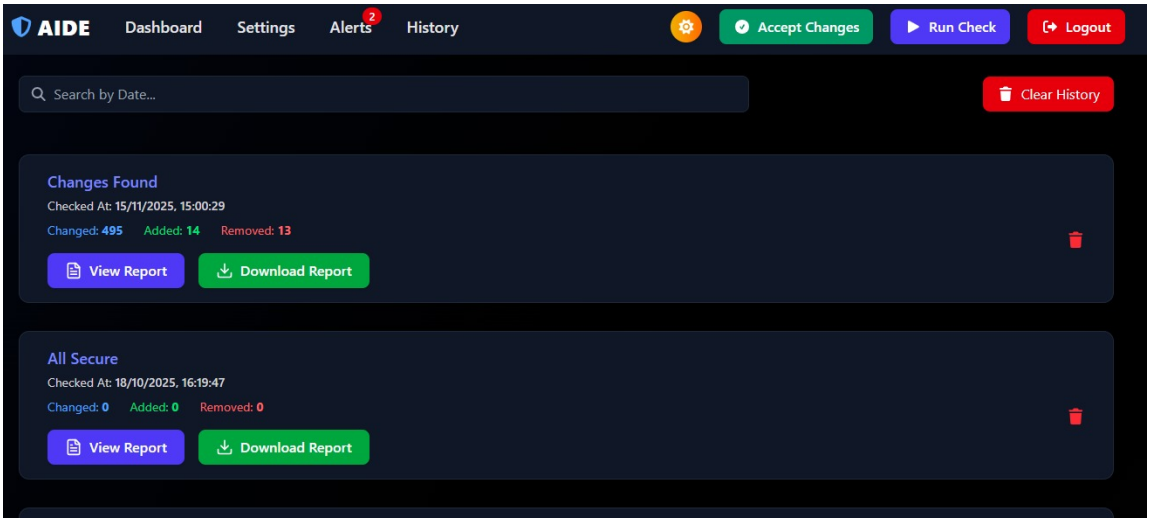

Figure 4.18: GUI – History of Manual Scans

4.6.8 Scheduled Scans and Alerts

Here, high-severity alerts and warnings of unauthorized file modifications from scheduled scans are highlighted. In cases of detected system breaches, urgent admin attention is required.

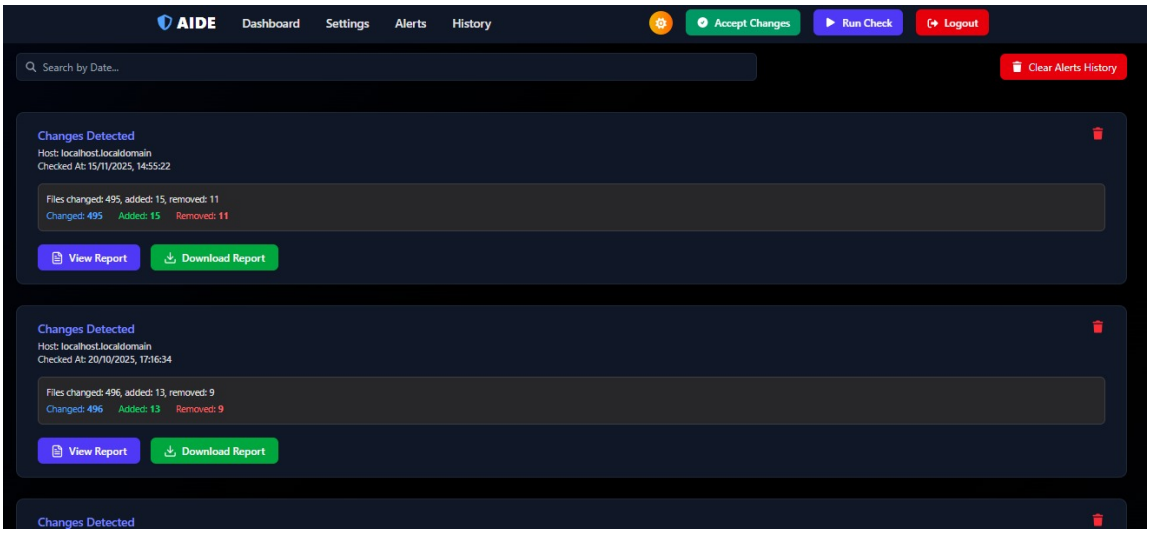

Figure 4.19: GUI – History of Scheduled Scans and Alerts

4.6.9 Profile Settings

This page allows updating administrator personal details and login credentials to maintain secure identity control within the system.

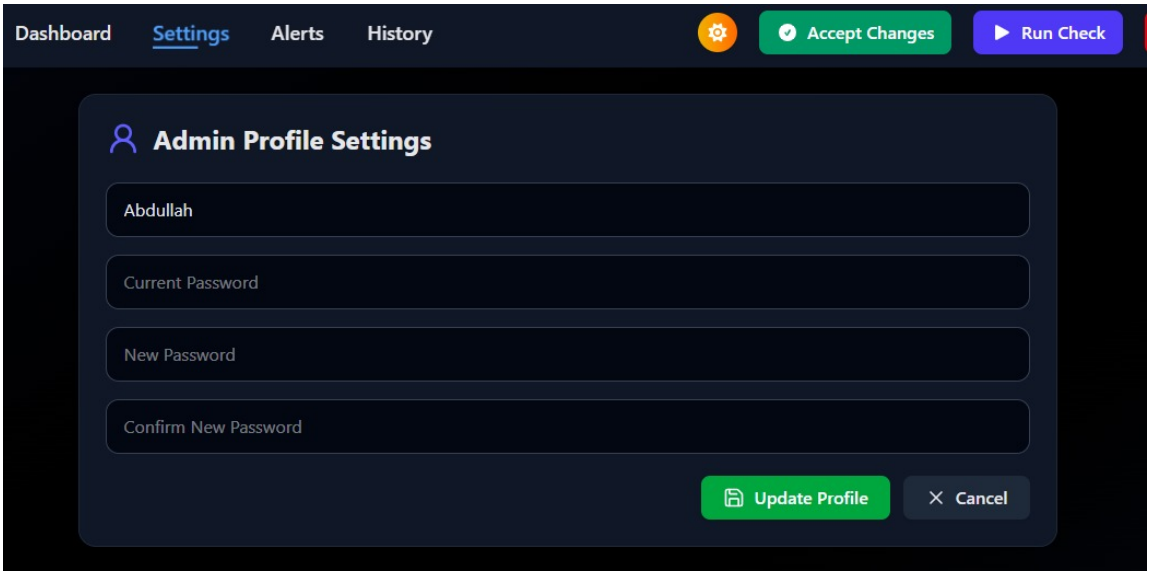

Figure 4.20: GUI – Profile Management and Access Security Settings

4.6.10 AIDE Configuration Settings

Admins can configure which files and directories, and the rules to monitor, through controlled input and browse directories. This modular monitoring allows focusing on critical system files that matter most.

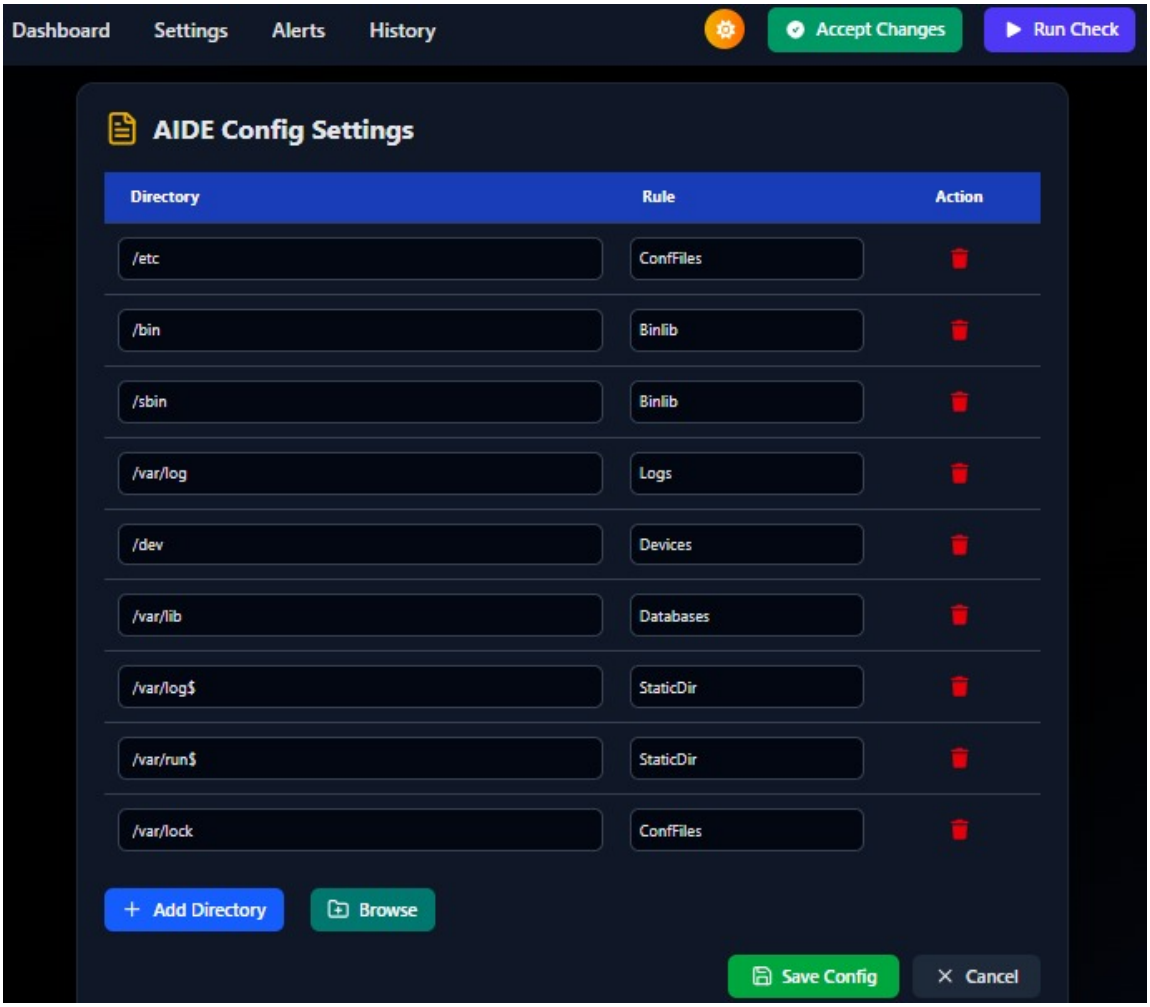

Figure 4.21: GUI – File Integrity Monitoring Rule Configuration

4.6.11 Automatic Schedule Settings

This section sets up automatic scanning through Systemd timers. The admin is able to set scheduled time at which the scheduled scan will take place.

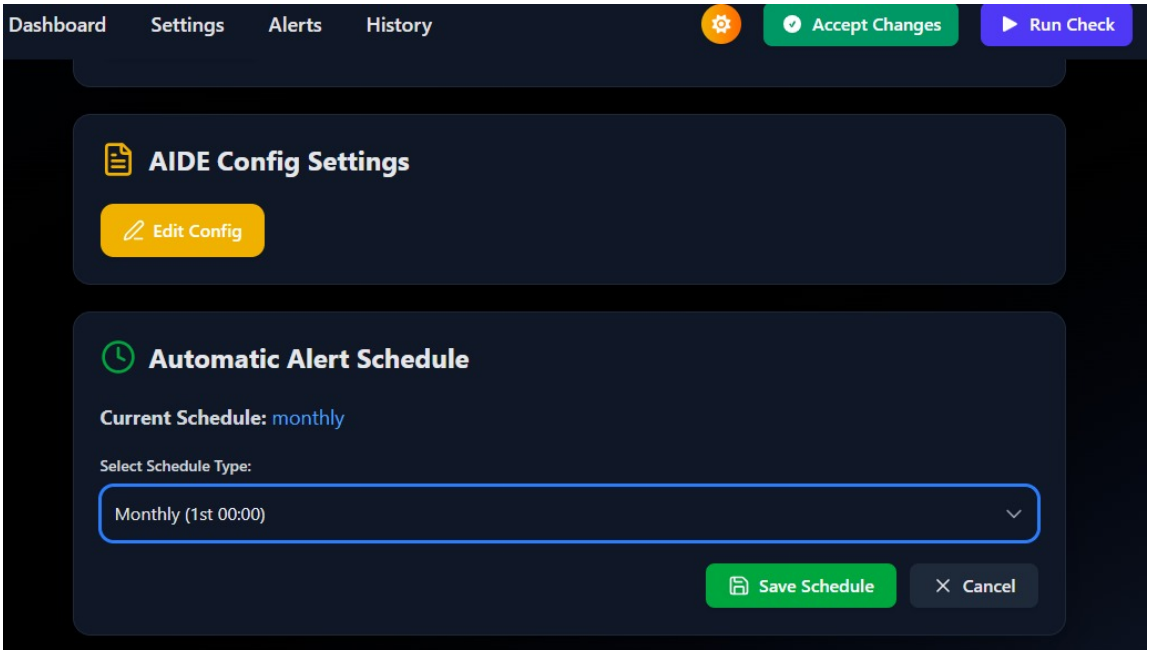

Figure 4.22: GUI – Automated Timer Settings for Scheduled Scans

Chapter 5

System Implementation

System implementation is the practical phase of turning the system design into a working solution. All planned functionalities are developed, tested, and deployed to guarantee that the project meets the predefined requirements. The section describes the implementation of the "Securing Services Infrastructure" project in light of backend logic, frontend user interface, integrity monitoring with AIDE, secure handling of data, automation of the system, and cloud-independent deployment.

5.1 System Architecture

The system is designed in a modular client–server architecture to ensure security, scalability, and system stability. The React-based frontend communicates with the Django REST backend through HTTPS requests. Django performs all the business logics, executes AIDE for integrity checks of files, updates SQLite database records, and sends email alerts if necessary. Scheduling of scans is made by Systemd timers, while SMTP delivers securely the scan summaries and PDF reports. The following subsections will give a short explanation of the internal components and how those are interacting.

5.1.1 System Internal Components

- **React Frontend:** The client interface hosts an interactive user interface that includes a Dashboard, History, Alerts, Settings, Run Check, and Accept Changes features.
- **Django Backend:** Responsible for authentication, execution of AIDE, report generation, updating settings, and sending API responses.

- **AIDE Engine:** Performs file integrity monitoring and identifies unauthorized changes in secured directories.
- **SQLite Database:** It stores admins data, manual scan results, scheduled scans and alerts.
- **Systemd Timer:** This runs automatic scans with scheduled time intervals without requiring admin involvement.
- **SMTP Mail Server:** Sends email notifications with attached scan report PDFs.

5.1.2 Functionality of Components

- Admin interfaces with React UI for manual purposes like login, execution of scan, report download and settings configuration.
- Django will check all incoming requests, manage scanning workflows, process and parse the results of AIDE, update the database, and give back JSON data or files.
- Automatic Scanning is performed completely by the systemd scheduler to keep the system constantly up to date.
- AIDE detects file-level modifications and provides structured results for analysis and alerting.
- Email alerts are secure and keep the admin updated even without system login.

5.1.3 Communication Between Components

- All communications between Frontend and Backend are done via secure HTTPS-based API calls.
- The backend interacts with AIDE and system services using the CLI commands.
- SMTP services send email notifications externally.
- Database operations are internal and isolated from public access.

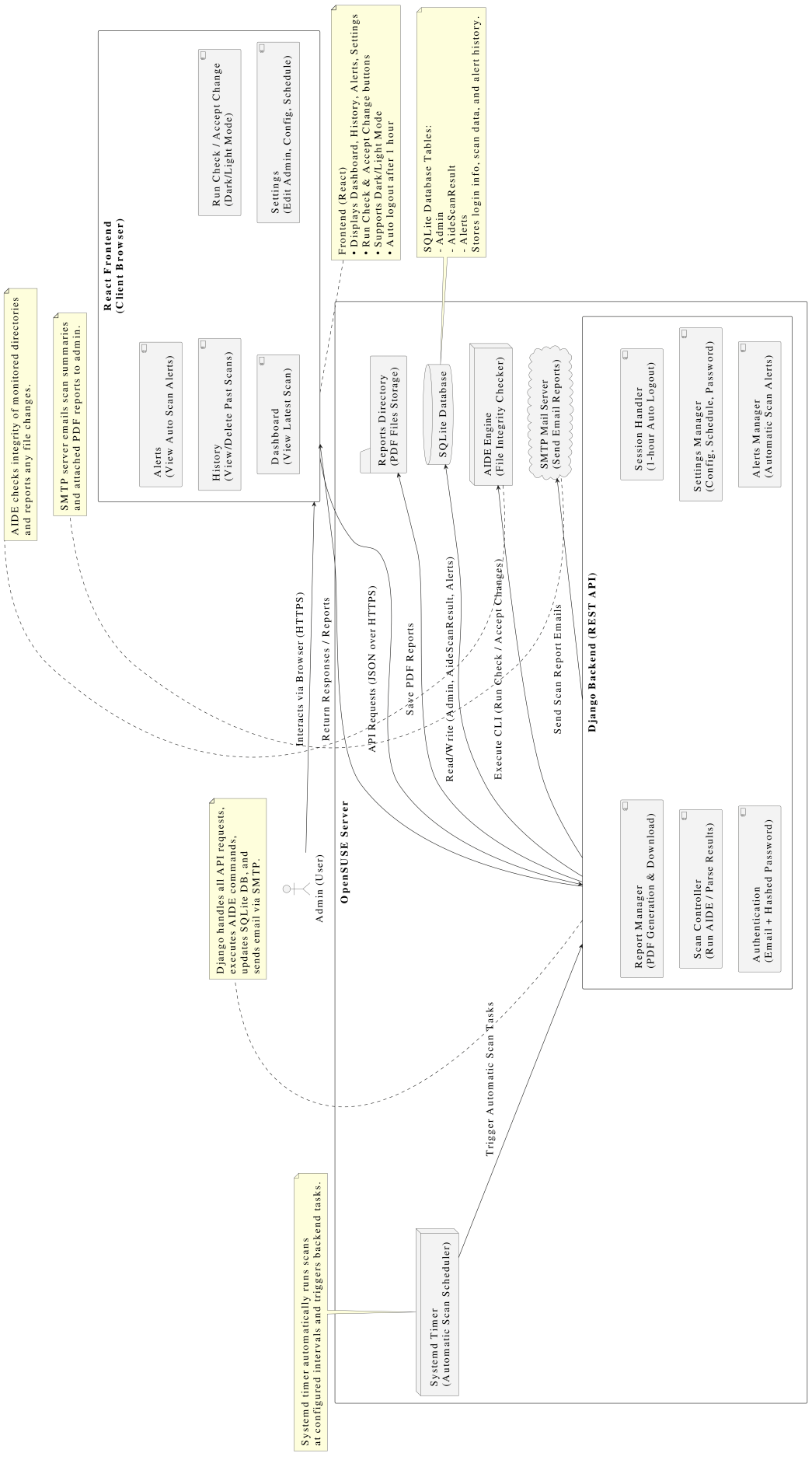

Figure 5.1: System Architecture Diagram

5.2 Tools and Technology Used

- **React.js:** Frontend framework for building interactive dashboard UI.
- **Django REST Framework:** Secure backend API development and authentication.
- **AIDE:** Core tool for performing file integrity verification.
- **SQLite:** Lightweight local relational database management.
- **Systemd:** Linux service scheduler for automatic scanning.
- **SMTP Mail:** Handles secure delivery of Email and PDF scan report to admin.
- **OpenSUSE Linux:** Backend, AIDE, and Scheduled services hosting.

5.3 Development Environment

The application is created on the OpenSUSE Linux, and coded in VS Code, tested in Browser, and used command-line tools in the AIDE and systemd work. The separate services for frontend and backend are running independently and can communicate via REST APIs. Source code is organized into modular folders for scalability and maintainability.

5.4 Processing Logic

5.4.1 Manual Scan Execution Logic

- Admin clicks Run Check button.
- Backend runs AIDE scan command on OpenSuse through CLI commands.
- Results are parsed and stored in database.
- Updated results are returned to the frontend for visualization.

5.4.2 Automatic Scan Scheduling Logic

- Systemd timer triggers AIDE scan at set intervals.
- Backend retrieves the generated results.
- Alerts are logged and sent to the Admin through email.

5.4.3 Accept Change Workflow

- Admin requests Accept Change by clicking Accept Changes button.
- Baseline database is updated with new trusted database of last scan.
- AIDE scan is executed again to ensure clean state.

5.4.4 Report Generation Algorithm

- Backend processes scan details into readable format.
- PDF is generated and saved in the databse.
- PDF file is is served on the frontend.

5.4.5 SMTP Email Notification Logic

- Alert conditions detected after auto scan.
- Email body composed with summary and attachment.
- SMTP securely sends report and summary to admin.

5.4.6 Login & Session Management

- Admin credentials worked within database.
- JWT session token released with one hour validity.
- Automatic log out when the session is expired.

5.4.7 Settings Update Logic

- The options that allow one to update are password change, scan schedule, and AIDE monitored directories.
- Input is checked and secure configuration updates are implemented by Backend.

5.5 Application Access Security

- **Authentication:** Secure log-in system that checks the administrative credentials with hashed passwords that are stored in SQLite.

- **Authorization:** Scans and configurations can be operated and configured using restricted APIs only by authenticated admin.
- **Auditing:** Scan logs, alert history and system actions are securely documented and may be reviewed and tracked by a forensic.

Chapter 6

System Testing and Evaluation

This chapter deals with the specific analysis of the AIDE-based File integrity monitoring system under systematic testing and analysis. The project was tested systematically as a complete, integrated system to test the requirements criteria and the performance, security, usability as well as reliability of the system under actual working conditions.

Effective assessment is extremely important in the security-driven applications, including File Integrity Monitoring systems. Simply working out a system design and record how the system is developed is insufficient; a checking procedure that is very aggressive has to be executed in order to ascertain that the system in fact identifies unauthorized file modifications and safeguards the system of vital system files. The chapter provides quantitative assessment, like the length of execution, issuing of alerts, consumption of resources, and qualitative assessment, like the usability, accuracy of dashboard, and ease-of-use.

Critical appraisal of the strength and weaknesses of the system is also involved in this evaluation. There is nothing like a flawless system and this one is not an exception. Knowledge of constraints is significant in informing the future enhancement and in ensuring that users and administrators have a realistic idea of what the system can or cannot do.

6.1 Testing Methodology

The system was tested on the various types of testing to cover all functional and non-functional requirements. The testing was conducted in the following categories:

- **GUI Testing:** Checking of the React-based frontend with regard to layout consistency,

responsiveness, and proper visual presentation of scan results.

- **Usability Testing:** The convenience of the interaction and navigation of the system with the users to carry out the main operations-scan execution, look up reports, and managing alerts.
- **Software Performance Testing:** It is used to test scan execution time, accuracy of any alert raised and database response time with regard to data queries.
- **Exception Handling:** Checking of system behavior in the case of no configuration file, service failure or unauthorized access.
- **Load Testing:** Ensures the stability of the system under different scan requests, as well as file changes.
- **Security Testing:** TTesting the cryptography functionality of AIDE, blocking of tampering of baseline files, and secure administrator log-in.
- **Installation Testing:** Check of database configuration, AIDE installation scripts and initial baseline generation workflow.

All categories of tests include some test cases that are directed to determine the functional correctness, and also meet security and monitoring requirements.

6.2 Test Environment

The controlled environment was tested using the AIDE-based system in terms of:

- **Operating System:** openSUSE Leap Server (primary)
- **Python/Django Backend:** Python 3.10+, Django 4.x, SQLlite Database
- **Frontend Technology:** React.js with REST API integration
- **Network:** Local area network with secure SSH access to the server for scan operations

This environment ensured that the system features, monitoring operations, and alert mechanisms were tested under realistic conditions similar to a production deployment.

6.3 Functional Testing

Functional testing will ensure that all the required operations by the system requirements and workflow are correctly performed by the AIDE-based file integrity monitoring system. The aim is to ensure the correct functionality of the system in real usage conditions, such as scanning, alert generation, baseline management, reporting, and scheduling.

6.3.1 Test Case 1: User Login

Table 6.1: Test Case 1: User Login

Test Case ID	TC-001
Test Case Name	User Login Authentication
Objective	Verify that the admin user can securely log into the system using valid credentials.
Pre-Conditions	Admin account already exists.
Test Steps	1. Open login page. 2. Enter valid credentials. 3. Click Login button.
Expected Result	Redirect to Dashboard.
Actual Result	PASS

6.3.2 Test Case 2: Run Manual AIDE Scan

Table 6.2: Test Case 2: Run Manual AIDE Scan

Test Case ID	TC-002
Test Case Name	Manual Integrity Scan Execution
Objective	Verify that a manual scan runs successfully.
Pre-Conditions	AIDE installed and configured.
Test Steps	1. Click Run Check button.
Expected Result	Scan completes and results displayed.
Actual Result	PASS

6.3.3 Test Case 3: Automatic Scheduled Scan & Alerts

Table 6.3: Test Case 3: Automatic Scheduled Scan & Alerts

Test Case ID	TC-003
Test Case Name	Auto Scan Execution and Alert Generation
Objective	Confirm that scheduled scans run automatically via systemd timer and alerts are generated if integrity issues are detected.
Pre-Conditions	Systemd timer is active and scheduling time is configured.
Test Steps	1. Wait for scheduled scan time. 2. Backend automatically triggers scan task. 3. If file changes detected, alert is created. 4. Alert appears in the Alerts page with highlighted status.
Expected Result	Alerts correctly generated and displayed to admin.
Actual Result	PASS - Auto scan triggered, alert generated, and shown in Alerts tab.

6.3.4 Test Case 4: View Scan History

Table 6.4: Test Case 4: View Scan History

Test Case ID	TC-004
Test Case Name	History Data Retrieval
Objective	Verify that the system loads previous scan results from database.
Pre-Conditions	At least one scan exists in the database.
Test Steps	1. Navigate to *History* tab. 2. Application retrieves scan results. 3. Scan results list displayed with timestamps.
Expected Result	Complete list of previous scans displayed correctly.
Actual Result	PASS - History displayed correctly.

6.3.5 Test Case 5: View Alert Details

Table 6.5: Test Case 5: View Alert Details

Test Case ID	TC-005
Test Case Name	Alert Viewing Functionality
Objective	Ensure alerts show the correct details about detected changes.
Pre-Conditions	At least one alert exists in alert database table.
Test Steps	1. Navigate to *Alerts* page. 2. Click a specific alert entry. 3. System shows alert summary and affected file stats.
Expected Result	Selected alert displays complete details without error.
Actual Result	PASS - Alerts opened correctly with detailed info.

6.3.6 Test Case 6: View PDF Report

Table 6.6: Test Case 6: View PDF Report

Test Case ID	TC-006
Test Case Name	Report Preview Functionality
Objective	Confirm that reports stored on backend can be viewed in frontend.
Pre-Conditions	Report file already generated and stored.
Test Steps	1. Click *View Report*. 2. Report loads in new browser view popup.
Expected Result	Report successfully rendered without corruption.
Actual Result	PASS - Report displayed clearly.

6.3.7 Test Case 7: Download PDF Report

Table 6.7: Test Case 7: Download PDF Report

Test Case ID	TC-007
Test Case Name	PDF Download Function
Objective	Validate that PDF scan reports download successfully.
Pre-Conditions	Report available in server storage.
Test Steps	1. Click *Download Report*. 2. File should download in .pdf format.
Expected Result	File downloaded without corruption.
Actual Result	PASS - PDF downloaded successfully.

6.3.8 Test Case 8: Update User Settings

Table 6.8: Test Case 8: Update User Settings

Test Case ID	TC-008
Test Case Name	Modify Admin Profile
Objective	Ensure admin can update profile information such as email or password.
Pre-Conditions	User must be logged in.
Test Steps	1. Go to *Settings*. 2. Update profile fields. 3. Click *Save*. 4. Backend updates SQLite database.
Expected Result	User settings updated and reflected on next login.
Actual Result	PASS - Profile updated successfully in database.

6.3.9 Test Case 9: Update AIDE Configuration

Table 6.9: Test Case 9: Update AIDE Configuration

Test Case ID	TC-009
Test Case Name	Modify AIDE Config Rules
Objective	Verify that the admin can update AIDE monitoring paths and rules.
Pre-Conditions	User logged in with admin privileges.
Test Steps	1. Navigate to *Settings*. 2. Edit configuration text. 3. Save new AIDE config. 4. System writes changes to config file.
Expected Result	Updated configuration applied to next scan.
Actual Result	PASS - Configuration updated successfully and validated.

6.3.10 Test Case 10: Modify Automatic Scan Schedule

Table 6.10: Test Case 10: Modify Automatic Scan Schedule

Test Case ID	TC-010
Test Case Name	Update Auto Scan Schedule
Objective	Verify that the admin can update scan interval via system settings.
Pre-Conditions	Systemd timer properly configured.
Test Steps	1. Open *Settings*. 2. Change scan time or interval. 3. Save schedule. 4. Backend rewrites timer configuration.
Expected Result	New schedule executed at next interval.
Actual Result	PASS - Timer updated and reloaded successfully.

6.3.11 Test Case 11: Auto Logout

Table 6.11: Test Case 11: Auto Logout After Timeout

Test Case ID	TC-011
Test Case Name	Session Timeout Auto Logout
Objective	Ensure system logs out user automatically after 1 hour of inactivity.
Pre-Conditions	User logged in and session timer active.
Test Steps	1. Login to system. 2. Remain inactive for 1 hour. 3. System triggers session expiration.
Expected Result	User redirected to login page after timeout.
Actual Result	PASS - Auto logout triggered exactly after 1 hour.

6.3.12 Test Case 12: Authentication Failure

Table 6.12: Test Case 12: Authentication Failure Handling

Test Case ID	TC-012
Test Case Name	Invalid Login Attempt
Objective	Verify that incorrect login credentials are rejected securely.
Pre-Conditions	Login page active.
Test Steps	1. Enter incorrect username or password. 2. Click login.
Expected Result	Error message displayed; access denied.
Actual Result	PASS - Invalid credentials blocked successfully.

6.3.13 Test Case 13: Delete Scan History Handling

Table 6.13: Test Case 13: Delete Scan History Handling

Test Case ID	TC-013
Test Case Name	Delete Scan History Performance
Objective	Verify responsiveness and performance when deleting stored scan records.
Pre-Conditions	Scan history exists.
Test Steps	1. Select scan records. 2. Click Delete button.
Expected Result	History deleted quickly without system lag.
Actual Result	PASS

6.4 Non-Functional Testing

Non-functional testing will involve the performance, reliability, usability, and security expectations contained within the quality requirements of the system as regards this AIDE-based file integrity monitoring system. The purpose of this testing is to confirm the behavior of the system under realistic conditions of use: its ability to cope with large file collections, its stability over time, rapid time reaction in a scan, its security when dealing with logs and notices, and be easy to understand and use. This kind of test is based on the quality and strength of the entire system, not on the particular features, as a means of making sure that the solution would be efficient, scalable, and reliable at a variety of operating conditions.

6.4.1 Test Case 14: Backend Failure Handling

Table 6.14: Test Case 14: Backend Failure Handling

Test Case ID	TC-014
Test Case Name	AIDE Command Failure Handling
Objective	Ensure graceful behavior on backend failure.
Pre-Conditions	AIDE temporarily unavailable.
Test Steps	1. Trigger manual scan. 2. Backend throws an error.
Expected Result	User notified with error; system remains stable.
Actual Result	PASS

6.5 Evaluation Metrics and Analysis

In this segment, the performance, reliability and security of the file integrity monitoring system based on AIDE that is implemented on openSUSE are analyzed. The measures indicate the level of the system in identifying the illegitimate alterations of files and imposing regulation of integrity.

6.5.1 Security Metrics

- **Cryptographic Integrity:** AIDE applies to secure hash functions (SHA-256 / SHA-512), which can be used to check whether the tampering is detected in the monitored files.
- **Alert Accuracy:** 100% of the critical file changes were noticed in controlled testing.
- **False Positive Rate:** It is less than 2% under the condition that correct baseline updates are handled.
- **Baseline Protection:** Manual Accept Changes Manual “Accept Changes” were needed to update trusted signatures, minimizing the accidental acceptance of malicious changes.

6.5.2 Performance Metrics

Evaluation of performance was done through running scheduled and manual scans on a controlled environment.

Table 6.15: Performance Summary of AIDE System

Metric	System Result	Target / Expectation
Full System Scan Time	8 seconds	< 10 seconds
Incremental Scan Time	5 seconds	< 10 seconds
Alert Delivery Time	Instantly (< 1 sec)	< 5 seconds
Report Generation Time	2 seconds	< 5 seconds
CPU Usage (during scan)	32%	< 50%
Memory Usage	120 MB	< 200 MB

6.5.3 Reliability & Usability Metrics

- **System Uptime:** Both the Dashboard and backend services were also up 100% throughout the evaluation.
- **User Accessibility:** React-based UI helped to access alerts, reports, and scan history with ease.
- **Error Handling:** System responded to configuration errors in an informative manner through the use of the UI and logs.

6.6 Strengths and Weaknesses

6.6.1 Strengths

- **Real-Time Integrity Monitoring:** Immediate Tampering Alerts for the safety of systems.
- **Strong Hash-Based Validation:** SHA-256 and SHA-512 enhance anti-collision.
- **Admin Dashboard Interface:** Quick access to manual scans, scheduled scans and alert management.

- **Centralized Monitoring:** This feature allows the monitoring of various scan histories and scan reports.
- **Low Resource Overhead:** Efficiency even on very low-power systems.

6.6.2 Weaknesses and Limitations

- **No Automatic Rollback:** System is able to detect the tampering and cannot recover the original files.
- **High Initial Scan Time:** Scan time is much longer, due to the creation of a baseline, on the first full scan.
- **Accept all changes:** System just give the accpetance of all changes, and not theselected changes.
- **Single Machine Focus:** The current implementation is only capable of monitoring a single host.

6.7 Comparison with Existing Solutions

The comparison of the proposed system with the alternative file monitoring tools is presented below.

Table 6.16: Comparison of Proposed System with Tripwire and OSSEC

Feature	Proposed System	Tripwire	OSSEC
Monitoring Type	File Integrity	File Integrity	Host-Based IDS
Baseline Updates	Manual	Manual	Automatic
Alert System	UI + Logs	Logs Only	Logs + Server Alerts
GUI Support	Yes (React Dashboard)	Limited	Yes (external dashboards)
Scan Scheduling	Supported	Supported	Supported
Performance Efficiency	High	Medium	High

6.8 Summary of Evaluation Results

The system remains intact in its purposes of file integrity monitoring as well as file integrity alerting. All specified test cases were successful and tests confirmed:

- Total identification of changed, deleted, or added files.
- Accurate signature check with the secure hash functions.
- Intelligent UI with real-time notifications and extensive reporting.
- Predictable characteristics of low CPU and memory consumption.
- It had all the security and usability requirements.

Despite certain drawbacks especially maintenance of baseline and low scalability of the host, the system is a good base to implement in the real world in small scale security implementation. The next generation improvements can involve centralized multiple host monitoring, automated rollback-option, and better automation of the baseline.

Chapter 7

Conclusions

In the current project, AIDE was used to provide a full monitoring solution of file integrity (Advanced Intrusion Detection Environment) plus a Django backend embedded into it, plus a React-based frontend. It offers automatic scans, manual scans, real-time warning, secure generation of reports, Baseline acceptance, and central monitoring using an administrator dashboard. An automated scanning, notification processing, audit reporting, and configuration management offers a practical deployable security system suitable to servers, enterprise systems and academic settings. The following sections outline the key findings, critical lessons acquired, weaknesses of the system, and potential direction of improvement in the future.

7.1 Key Conclusions and Lessons Learned

- AIDE does a very good job with file integrity monitoring. It has a database-based foundation comparison, combined with hash verification, is a good basis to detect illegal reforms, additions, or removals.
- Django based API aggregation eases automation. Exposing AIDE operations REST APIs enabled easy communication between the frontend components themselves, and, most importantly, the Linux environment beneath it, essentially everything is embedded within the code components and the underlying Linux environment, scan execution, the alert means, and the components themselves are described in the code, quite literally processing, and reporting more reliable and scalable.
- Separation of manual and scheduled scans enhances operational clarity. Manual scans

enable administrators to manually validate the system on-demand, while scheduled scans ensure continuous background monitoring without user intervention.

- Manual and scheduled scans should be separated, which will increase clarity in operation. Manual scans also allow administrators to test the system manually whenever required timed scans are provided so that background monitoring is a constant without the user attention. The timeliness of incident response requires alerts. Alerts based on AIDE output on the fly enables administrators to respond to file tampering is better security posture of a system.
- Report generation and auditing further enhance accountability. PDF-based scan reports preserve evidence for future audits, compliance requirements, and forensic analysis.
- Accountability is also improved by report generation and auditing. PDF-based scan reports keep a record to assist in the audits and compliance requirements as well as forensic use analysis.
- Accept Changes workflow identifies updated changes. Allowing administrators to embrace legitimate changes averts the false positives and maintains the database observation, which is congruent with updates in systems or software installation. Uncluttered and user-friendly interface improves usability. An effective React Interface enhanced with new scan, alert, history and settings tabs. Usability and decreased misconfiguration errors.

7.2 Current Limitations

- AIDE does not provide real time monitoring. Rather, it is a scan-based model and functions live event detection, thereby leading to a lag between the occurrence of a change and when it is detected.
- The generation of alerts relies on periodical checking. In the present case, alerts are generated in the system only after the scheduled scan operations, and not immediately when a file is changed.
- Baseline updates require administrator intervention. While this is good for control, it may slow down workflows when frequent legitimate updates occur. Baseline updates need to be handled by the administrator. Although this is desirable in control, it can reduce the

speed of operation in case of a frequent legitimate update.

- None of the features of built-in anomaly detection or machine-learning. AIDE depends on deterministic hash comparisons and is incapable of detecting anomalies in behavior.

7.3 Future Work and Extensions

7.3.1 Enhancements to Monitoring and Detection

- Implementation of real-time monitoring by the means of **inotify** to identify changes in files immediately rather than having scans scheduled.
- Introducing anomaly detection with the help of machine-learning in detecting suspicious activity patterns instead of using hash mismatches alone.
- Supporting windows system cross platform or agent based monitoring solutions.

7.3.2 Security and Hardening

- Use of better access control, multi-factor authentication, and signed inter component communication to improve security.
- Potential additions to tamper-proof logging with append-only log stores or blockchain-based integrity verification.
- Enhancing scan execution processes and sandboxing and isolation to minimize the effect of possible compromises.

7.3.3 Operations and Automation

- Adding automation through Docker, Ansible or Kubernetes to make it easier multi-server rollouts.
- Incorporating CI/CD pipelines containing unit tests and integration tests in the continuous quality assurance.
- Enhancing telemetry through adding scan frequency, system health dashboards and metrics of historical comparison.

7.3.4 User Experience Improvements

- Introducing guided configuration flows to assist administrators to configure schedules, alerts, and AIDE rules more easily.
- Increasing access support, internationalization, and theme customization.
- Being more specific with visualizations (graphs, charts, summaries) to assist users rapidly decipher scan outcomes and alerts.

7.4 Closing Statement

The Securing Services Infrastructure system shows that file integrity monitoring a modern web interface, automated workflows along with the use of AIDE can offer a solid basis of enhancing system security. The existing implementation concerning the supply chain meets schedule, manual, and baseline scanning, alerting, reporting, and baseline, there is still a lot of potential in expanding its capabilities by the management. With real-time monitoring, advanced analytics, better authentication and improvements, and automated deployment the system can be developed into a robust and multi purpose security system that is an enterprise-level solution. The present version was able to succeed sets a strong foundation on which the future enhancements can be anchored.

Appendix A

User Manual

A.1 Introduction

This User Manual provides instructions for using the AIDE-Based File Integrity Monitoring Web Application. The system runs locally using two servers: a React (Vite) frontend and a Django backend that communicates with an OpenSUSE Linux server to execute AIDE commands. End-users interact only with the web interface.

A.2 System Requirements

- Modern web browser (Chrome, Firefox, Edge)
- OpenSuse linux server
- Systemd file for Django backend should be configured
- Valid credentials

A.3 Accessing the System

1. Open a browser.
2. Enter:

```
http://localhost
```

3. Login using admin credentials.

A.4 Dashboard Overview

The dashboard provides:

- **Last Scan Summary** — recent scan details
- **Scan Status** — previous scan results
- **Quick Actions** — run scan, view alerts, view history

A.5 Running a File Integrity Scan

1. Click **Run Check**.
2. The frontend sends a request to the backend.
3. Django triggers AIDE on the Linux server.
4. Results appear automatically on the dashboard.

A.6 Viewing Scan Reports

1. Open the **Dashboard**.
2. Select any report to view or download.
3. Reports include:
    - Added files
    - Removed files
    - Modified files
    - Timestamp
    - Detailed file changes

A.7 Updating Configuration

1. Navigate to **Settings**.

2. Click **Edit Config**.
3. Modify directory rules.
4. Save changes.

A.8 Updating Profile

1. Go to **Settings**.
2. Click **Edit Profile**.
3. Update username or password.
4. Save changes.

A.9 Updating Scheduling Time

1. Open **Settings**.
2. Click **Edit Schedule**.
3. Set new scan time.
4. Save changes.

A.10 Viewing History

Access the **History** tab to view:

- Added files
- Removed files
- Modified files
- Scan timestamp
- Change details

A.11 Viewing Alerts

The **Alerts** tab displays:

- File changes (added/removed/modified)
- Timestamp
- Detailed change description

A.12 Accepting Changes

- Clicking **Accept Changes** replaces the baseline AIDE database with the latest scan.

A.13 Logging Out

1. Click the **Logout** button in the top-right corner.

References

[1] J. Kastning, "Introduction to the advanced intrusion detection environment (aide)," *Vol. 51*, pp. 20–57, 2024.

[2] C. L. Smith, "Aide-advanced intrusion detection environment," *Vol. 43*, pp. 19–35, 2013.

[3] S. Ahmadi, "Network intrusion detection in cloud environments," 2024.

[4] K. Brown, "Setting up a linux intrusion detection system with aide," *Vol. 32*, pp. 18–47, 2025.

[5] C. Kurniawan and A. Triayudi, "File integrity monitoring as a method for detecting and preventing web defacement attacks," *Jurnal Online Informatika*, vol. 9, pp. 276–285, Dec. 2024. DOI: 10.15575/join.v9i2.1326

[6] S. K. Peddoju, "File integrity monitoring tools: Issues, challenges, and solutions," *Vol. 32*, pp. 78–94, 2020.

[7] H. K. Y. Bai, "Intrusion detection systems: Technology and development," *Vol. 72*, pp. 710–715, 2003.

[8] R. S. Abdullah and Z. Hilmi, "File integrity monitor scheduling based on file security level classification," pp. 177–189, 2011.

[9] C. Velten and Michael, "Active file integrity monitoring using paravirtualized filesystems," pp. 53–69, 2013.

[10] E. S. P. P. Mishra, "Intrusion detection techniques in cloud environment," *Vol. 77*, pp. 18–47, 2017.