CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Monitoring and Alert System for Shared Computer Laboratories

Bachelor thesis in Computer Science and Engineering

Sukaina Akar
Angelika Hagberg
Houmam Kadamani
Joel Moberg
Sufian Tariq

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Bachelor's thesis 2026

A Monitoring and Alert System for Shared Computer Laboratories

Sukaina Akar
Angelika Hagberg
Houmam Kadamani
Joel Moberg
Sufian Tariq

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

A Monitoring and Alert System for Shared Computer Laboratories

Supervisor (Handledare): Ilias Chanis, Department of Computer Science and Engineering
Examiner: Arne Linde, Department of Computer Science and Engineering
Graded by teacher (Medrättande lärare): Örjan Askerdal, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg, Sweden
Telephone +46 31 772 1000

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Shared computer laboratories at universities provide essential resources to students, but they are also vulnerable to theft of internal hardware components, such as graphics cards, memory modules, and storage devices. Detecting such thefts is challenging because many benign events, including maintenance, configuration changes, and temporary network issues, can produce symptoms similar to physical tampering. This thesis presents the design, implementation, and evaluation of a prototype software-based monitoring and alert system for shared computer workstations. The system uses a client-server architecture in which lightweight clients installed on laboratory computers send periodic heartbeats and provide information to the server. The server applies context-aware alarm logic to classify events and reduce false alarms. The system was evaluated through scenario-based testing in a laboratory environment. The results show that the system reliably detects hardware component changes and that the context-aware alarm logic reduces false alarms compared to an approach in which every unavailability event triggers an alert. The system was able to distinguish between benign and suspicious events in evaluated scenarios where contextual evidence was available. However, certain situations, such as a single machine losing power, remain inherently ambiguous, as they produce system-level signals identical to those caused by physical tampering.

Sammanfattning

Datorlaborationssalar vid universitet utgör viktiga resurser för studenter, men de är också sårbara för stöld av interna hårdvarukomponenter som grafikkort, minnesmoduler och lagringsenheter. Att upptäcka sådana stölder är en utmaning eftersom många legitima händelser, inklusive underhåll, konfigurationsändringar och tillfälliga nätverksproblem, kan ge upphov till symptom som liknar fysisk manipulation. Denna rapport presenterar design, implementering och utvärdering av ett mjukvarubaserat övervaknings- och larmsystem för datorer i laborationssalar. Systemet använder en klient-server-arkitektur där klienter installerade på datorer skickar periodiska signaler och information till servern. Servern tillämpar kontextmedveten larmlogik för att klassificera händelser och minska falsklarm. Systemet utvärderades genom scenariobaserad testning i en datorlaborationssal. Resultaten visar att systemet på ett tillförlitligt sätt upptäcker förändringar i hårdvarukomponenter och att den kontextmedvetna larmlogiken minskar antalet falsklarm jämfört med ett tillvägagångssätt där varje fall av otillgänglighet genererar ett larm. Systemet kunde skilja mellan legitima och misstänkta händelser i de utvärderade scenarier där kontextuell information fanns tillgänglig. Resultaten belyser dock att vissa händelser, såsom att en enskild dator förlorar strömförsörjningen, förblir svårtolkade eftersom dessa händelser ger upphov till signaler som är identiska med de som orsakas av fysisk manipulation.

Contents

1
Introduction

Monitoring large numbers of computers connected to a network presents significant challenges in interpreting system behavior and distinguishing between normal and anomalous events. In recent years, Chalmers University of Technology has experienced recurring theft incidents in its computer laboratories, where components such as memory modules, graphics cards, mice, keyboards, and storage devices have been stolen from stationary computers [1]. Similar incidents have also been reported in other university environments. For example, a computer laboratory at Princeton University experienced the theft of multiple graphics processing units (GPUs) [2], demonstrating that internal components in shared computer environments can be targeted. These incidents make it difficult to maintain computer availability. In many cases, theft can only be discovered when a user or technician notices degraded performance or a missing component, which may occur days or weeks after the incident.

These challenges motivate the need for a software-based solution capable of detecting component-level theft on stationary computers and notifying relevant personnel when an incident occurs. Such a system should be cost-effective to implement and easy to maintain. It should also minimize network congestion and false alarms, while providing timely notifications.

1.1 Problem

In shared computer laboratories at Chalmers University, a large number of computers are accessible to students. This creates challenges for ensuring the physical security of internal hardware components.

The problem might initially appear to be limited to detecting unavailable devices. However, deeper analysis reveals that the problem is more complex, as many benign events produce signals similar to those caused by hardware removal. As a consequence, a simple availability monitoring system cannot distinguish between normal events and a security breach. This would lead to a system that produces ambiguous results and has a high risk of false alarms. The primary challenge is therefore not only to detect that a computer is unavailable but also to interpret the underlying cause of unavailability. This requires combining information from individual devices with contextual patterns across multiple systems to distinguish between different types of incidents.

1.2 Aim

The aim of this project is to design and implement a prototype software-based monitoring and alert system for shared computer workstations. The system will monitor workstation availability and hardware state, detect anomalies such as disconnections or hardware changes, and provide notifications and a status overview to support operators. Furthermore, the project evaluates whether a lightweight, context-aware solution can distinguish between benign events and potentially suspicious incidents using rule-based logic and existing monitoring data. It also examines whether this can be achieved while maintaining a low false alarm rate. The expected outcome is a functional prototype together with an evaluation of its effectiveness in controlled laboratory environments.

1.3 Research Question

Based on the identified problem, the main research question is formulated as follows: *How can the cause of computer unavailability be interpreted, using available system-level data and contextual information, in order to distinguish benign events from potentially suspicious physical intervention while maintaining a low false alarm rate?*

1.4 Scope

This project focuses on software-based monitoring of computer workstations in shared laboratory environments. The system is designed specifically for Windows-based machines, reflecting the typical configuration in the target environment. The outcome of this project is a research prototype rather than a production-ready system. Full-scale deployment, including integration with institutional IT infrastructure and long-term operational use, is outside the scope of this work. Physical security solutions, such as camera-based surveillance, presence sensors, or alarm systems, are not considered, as the focus of this work is software design and system integration. These design choices enable the work to focus on interpreting system behavior and detecting hardware-related anomalies within a controlled environment.

2

Related Work

This chapter reviews existing approaches to computer monitoring and theft prevention. The solutions are grouped into four categories and evaluated based on their relevance to this project.

2.1 GPS-Based Security Systems

Some monitoring systems employ GPS-based tracking, particularly in laptops and mobile devices, to determine the physical location of monitored equipment. These systems rely on embedded GPS functionality to retrieve geographic coordinates, which are then compared against constructed virtual geofencing boundaries [3]. If a device moves beyond a predefined zone, it is interpreted as a potential security event and may trigger an alarm or tracking response. An example of such a security system is Prey [4]. It triggers an alert primarily when a computer is lost or out of reach, suggesting possible theft.

Although this approach may be effective for portable devices, it is less suitable for stationary computers in laboratory environments, where devices are rarely moved and theft more commonly involves the removal of individual hardware components rather than entire machines. While Prey can generate alerts when a device becomes unreachable or goes offline, such events may equally result from technical issues or network interruptions, leading to a higher rate of false alarms. In environments with hundreds or thousands of machines, this lack of specificity significantly reduces the usefulness of the system.

2.2 Software-Based Monitoring Systems

Lansweeper is an administrative monitoring tool that collects and stores information about computers within an organization [5]. It collects data from multiple sources, including Windows Management Instrumentation (WMI), to retrieve information about Windows systems. The platform is capable of detecting changes in workstation hardware and reporting these changes to administrators.

Lansweeper is suitable for detecting hardware changes, but is not intended to function as a theft detection system. Notifications are delivered primarily via email and

can be configured as scheduled report alerts or event-based alerts triggered by detected system events. This is not ideal for real-time incident response, as email-based alerts may be delayed or overlooked. Furthermore, Lansweeper provides extensive administrative functionality that is not required for this specific use case. The platform does not distinguish between normal and forced shutdowns, nor is it capable of detecting the disconnection of peripherals such as mice or keyboards.

Another monitoring platform is N-central, developed by N-able [6]. It is a system management and monitoring solution designed for servers and workstations, supporting operating systems such as Windows, macOS, and certain Linux distributions. The system consists of a central server, a management dashboard, and clients installed on individual devices.

The primary focus of N-central is to monitor system performance, including both hardware and software status, in order to detect issues that may affect productivity. By identifying performance degradation early, the system can help reduce downtime and minimize the need for manual intervention from IT personnel. This enables a more proactive approach to system maintenance and provides IT departments with improved oversight of their infrastructure.

However, similar to Lansweeper, N-central is not specifically designed to detect physical changes to hardware components. As a result, it does not have the capability to identify component-level theft or to distinguish such events from other types of system anomalies.

2.3 Heartbeat-Based Health Monitoring

Heartbeat mechanisms are widely used in computer systems to monitor the availability of networked nodes [7]. A heartbeat typically consists of periodic messages sent by a node to indicate that it is operational.

While this mechanism is effective for detecting that a node has become unreachable, it does not inherently provide information about the cause of the failure. A missed heartbeat may indicate hardware failure, network disruption, administrative shutdown, or physical manipulation. The system proposed in this project extends the heartbeat concept by incorporating contextual interpretation logic that attempts to classify the reason for unavailability rather than simply reporting it.

2.4 Physical Security and Alternative Approaches

An alternative approach to theft prevention is the use of physical security measures, such as camera-based monitoring. Surveillance systems can deter theft and provide evidence after an incident has occurred. However, this approach introduces several significant disadvantages. Continuous camera monitoring raises privacy concerns and requires compliance with data protection regulations. Additionally, effective use of surveillance systems often requires active monitoring by security personnel,

which increases staffing requirements and operational costs. Installation and maintenance of cameras, cabling, storage systems, and monitoring infrastructure also result in high initial and long-term costs [8]. Automated monitoring using artificial intelligence and image recognition could reduce the need for human supervision. However, in addition to the aforementioned data protection and privacy concerns, such systems would require a large amount of computational resources for developing, training, and maintaining. This would significantly increase system complexity and implementation cost.

2.5 Summary and Identified Gap

The existing approaches discussed above address different aspects of computer security, but none are specifically designed to detect component-level theft in desktop computer laboratories. GPS-based systems focus on device-level tracking, administrative tools detect hardware changes, and heartbeat-based monitoring identifies system unavailability. However, these approaches do not combine different types of system information and cannot distinguish between benign and suspicious events.

This highlights the absence of a system that combines lightweight hardware and availability monitoring with contextual interpretation logic to classify the likely cause of workstation unavailability. The system proposed in this project addresses this gap by applying rule-based classification to standard monitoring signals using contextual information.

3

Methodology

This chapter describes the methodological approach used to design, implement, and evaluate a prototype monitoring and alert system for computer workstations. The primary outcome is a functional prototype, which is assessed through systematic testing and evaluation to determine whether the prototype can interpret workstation unavailability and reduce the likelihood of false alarms. The chapter covers the research and development approach, project phases, evaluation methodology, evaluation criteria, and data collection methods.

3.1 Research and Development Approach

The prototype development followed an iterative prototyping approach inspired by Agile development principles [9]. Instead of fully specifying and implementing the system in a single pass, development was organized around building a minimum viable product (MVP) that was then refined through repeated cycles of implementation, testing, and adjustment. In practice, this meant that the project was organized in short development cycles of one or two weeks, each ending with a review of what had been implemented and what should be prioritized next. No formal Agile framework was adopted; however, the iterative structure reflects core Agile values.

This approach was chosen for several reasons. The alarm logic depends on threshold values and contextual rules that cannot be fully specified in advance. Instead, they require empirical calibration through testing. An iterative approach also allows for early identification of technical challenges that may influence subsequent design decisions. Furthermore, working in short iterations made it possible to maintain a functional prototype throughout the project, reducing the risk of integration failures later in the development process.

The iterative approach was implemented through an incremental implementation strategy, in which core monitoring functionality such as heartbeat and basic status detection was implemented before more advanced features such as interpretation logic and alert mechanisms were added. Continuous validation was applied, with each iteration including testing to assess system performance.

3.1.1 Considered Alternatives

A plan driven (waterfall) approach [10, p. 9] was considered but deemed unsuitable for this project. A sequential approach where requirements, design, and implementation are completed in strict order would have made it difficult to incorporate findings from testing back in the design. This would limit the opportunity to adjust the alarm logic based on empirical observation.

3.2 Project Phases

The project was carried out over approximately 18 weeks and was divided into four main phases with different focus areas. These focus areas are presented in approximate chronological order, but they should not be understood as strict sequential phases. In practice, implementation, testing, refinement, and report writing overlapped throughout the project.

1. **Research and Planning:** This phase focused on understanding the problem domain, reviewing related work and formulating the research question. Requirements were gathered by reviewing related literature and discussions with relevant stakeholders. The system architecture was sketched out at a high level.

2. **Core Implementation:** The first prototype was developed during this phase. This included the heartbeat module, server logic and basic client-server communication. By the end of this phase, the system could receive heartbeats and store them with timestamps in the database.

3. **Alarm Logic, Refinement and Writing:** The context-aware alarm logic was implemented and iteratively refined. The writing of the final report began alongside refinements and additions to the system.

4. **Evaluation and Writing:** The final prototype was evaluated through scenario-based functional tests. The results were analyzed and the report was finished.

3.3 Evaluation Methodology

The prototype was primarily evaluated using scenario-based functional testing. This approach involves defining a set of test scenarios that represent situations that the system is expected to handle. The evaluation focuses on the system's ability to correctly classify workstation states based on heartbeat signals and contextual indicators.

Before each testing scenario, a description of the simulated event was defined. The expected system behavior and the conditions under which the test was considered passed were specified in advance. The scenarios were executed manually under controlled conditions, meaning that only the variable under investigation was changed

while other conditions remained stable. Each scenario was repeated to ensure consistent behavior. The specific scenarios and their results are presented in Section 5.2.

3.3.1 Evaluation Criteria and Metrics

To assess the performance of the prototype, three evaluation criteria were defined: detection accuracy, false alarm rate, and detection latency.

Detection accuracy refers to whether the system correctly identifies suspicious unavailability events. This was assessed by checking whether the system triggered an alert in scenarios classified as suspicious and whether it avoided triggering alerts in scenarios classified as benign.

False alarm rate refers to the proportion of alerts generated by the system that corresponded to benign events rather than genuinely suspicious events. This criterion was used to evaluate whether the prototype could avoid unnecessary alerts during normal or non-suspicious situations.

Detection latency refers to the elapsed time between the occurrence of a suspicious event and the system generating an alert or notification. Lower latency is generally desirable, but must be balanced against the risk of false alarms caused by premature detection.

3.4 Data Collection

Data was collected through the system's own logging mechanisms. The monitoring system records relevant system events, including heartbeat messages, hardware state changes, and generated alerts, each associated with a timestamp. These logs form the primary data source used for evaluation. During the scenario-based testing, the tester also recorded the exact time and nature of the simulated event manually, creating a ground truth log that could be compared against the system's log to verify correctness and measure detection latency.

4

System Design

This chapter describes the design of the prototype monitoring and alert system, including its overall architecture, the roles of the client and server components, the data flow between them, the design of the alert logic, and the graphical user interface.

The prototype follows a client-server architecture in which lightweight monitoring clients are deployed on workstations in a computer laboratory and communicate periodically with a centralized server [11]. This architectural design allows monitoring data from multiple machines to be collected in one place, enabling a centralized evaluation of workstation status.

At a high level, each client sends a heartbeat message to indicate that the workstation is active. In addition, the client reports hardware and peripheral information that can be used to detect changes in workstation configuration. The server receives monitoring data from clients, maintains machine state, and exposes alert information through API endpoints.

Figure 4.1 gives an overview of this architecture by showing how the workstation clients, backend server, database, operator interface, and Wake-on-LAN (WoL) recovery mechanism interact. The figure provides the structural context for the more detailed client-side and server-side descriptions that follow.

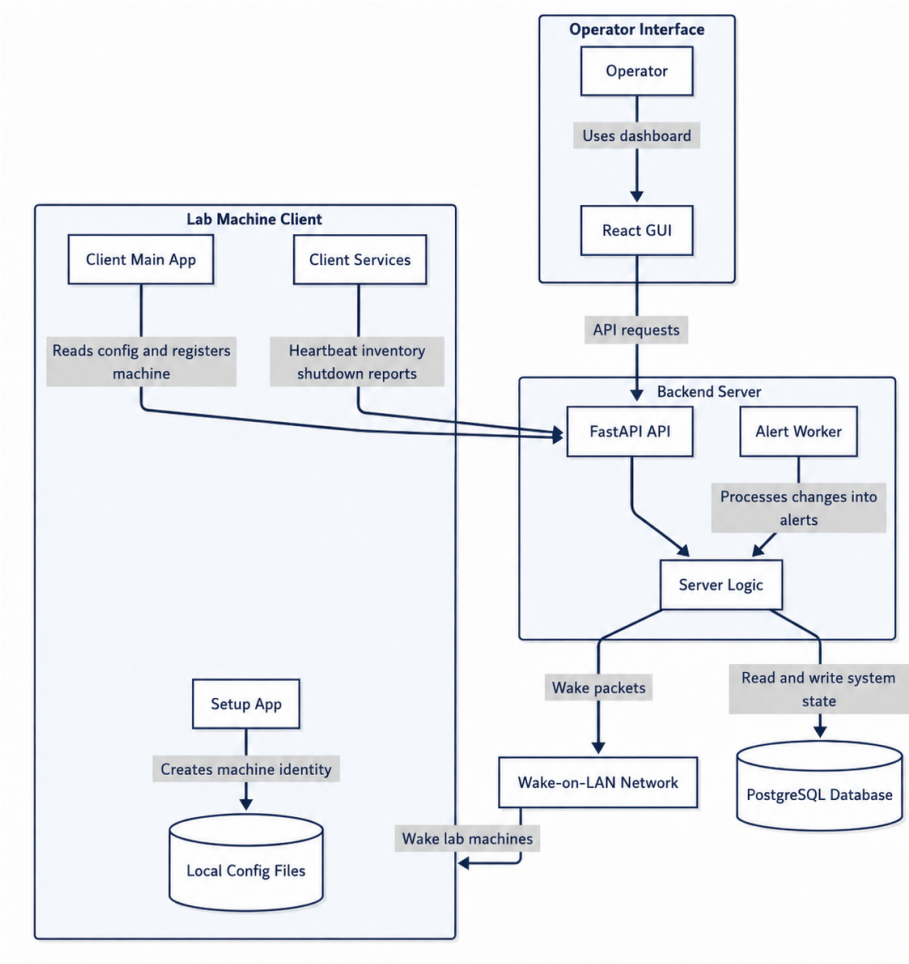

Figure 4.1: Overview of the monitoring and alert system architecture, showing the interaction between laboratory machine clients, the backend server, the database, the operator interface, and Wake-on-LAN recovery.

4.1 Design Goals

The system design is guided by the following goals:

- **Low false alarm rate:** Alarms must be reliable enough so that operators do not learn to ignore them.
- **Interpretability:** Alarms should be explainable, allowing operators to understand why an alert was generated.
- **Scalability:** The system should handle monitoring of many workstations without excessive network traffic or manual administration.
- **Robustness to common failures:** The design must tolerate temporary network issues, reboots, sleep states, and planned maintenance.
- **Minimal user impact:** A fundamental design principle is to keep the client lightweight so that it does not interfere with normal workstation use.

These design goals were derived from the project specification and refined through discussions during the development process as the practical requirements of the system became clearer.

4.2 Technology and Implementation Choices

The design and implementation of the system required selecting appropriate technologies for both the client and server components. These choices were guided by the system's design goals, particularly scalability, reliability, maintainability, and low resource usage.

4.2.1 Client Technology Choice (Go)

The client component was implemented in Go [12]. Go is an open-source programming language designed for building efficient and reliable software. This choice was primarily motivated by Go's ability to produce lightweight, statically compiled binaries with minimal runtime dependencies. Since the client is deployed as a background service on multiple workstations, it is important that it consumes minimal system resources and does not interfere with normal usage.

Go also provides strong support for concurrency through lightweight goroutines, which simplifies the implementation of periodic tasks such as heartbeat transmission and hardware monitoring. Additionally, the ability to compile the client into a single executable simplifies deployment and installation across multiple machines.

4.2.2 Server Implementation (Python + FastAPI)

The server component was implemented in Python using the FastAPI framework [13]. This choice was motivated by FastAPI's support for rapid development, clear API design, and strong integration with modern Python tooling.

FastAPI provides automatic request validation, asynchronous request handling, and built-in documentation generation, which simplifies the implementation of API endpoints. These features are particularly useful in a system where multiple clients communicate frequently with a central server.

Python was chosen for its flexibility and simplicity, which facilitated rapid prototyping and iterative development, aligning with the project's development approach described in Section 3.1.

4.2.3 Database (PostgreSQL)

PostgreSQL [14] was selected as the database system due to its reliability, strong support for relational data, and advanced querying capabilities. The system requires structured storage of machine states, historical events, and alert data, making a relational database a suitable choice.

PostgreSQL also supports transactions and efficient indexing, which are important for ensuring consistency and performance as the number of monitored machines increases [15, 16].

4.2.4 ORM and Migrations (SQLAlchemy + Alembic)

SQLAlchemy [17] was used as the Object-Relational Mapping (ORM) tool to manage interactions between the application and the database. This abstraction simplifies database operations and improves code maintainability by allowing developers to work with Python objects instead of raw SQL queries.

Alembic [18] was used for database schema migrations. This enables controlled evolution of the database structure over time, which is particularly important in an iterative development process where the data model may change. Without a migration tool, schema updates would need to be handled manually, increasing the risk of inconsistencies and errors.

4.2.5 Summary of Implementation Choices

Overall, the selected technologies reflect a trade-off between performance, ease of development, and maintainability. Lightweight and efficient tools were prioritized for the client, while flexibility and rapid development were prioritized for the server. This combination aligns with the system's architecture, where computational complexity is centralized on the server while keeping the client minimal.

4.3 Client Architecture

As shown in Figure 4.2, the client is separated into three different modules: the heartbeat module, the hardware inventory module and the shutdown notification module. A complementary heartbeat-once module is also used to transmit heartbeats during sleep mode. All modules except heartbeat-once are implemented as Windows services that starts automatically when the operating system boots and runs continuously in the background [19]. Its responsibilities are limited to data collection and transmission, and it does not perform any alarm evaluation on its own. This separation of concerns ensures that the client remains simple and that all analytical logic is on the server.

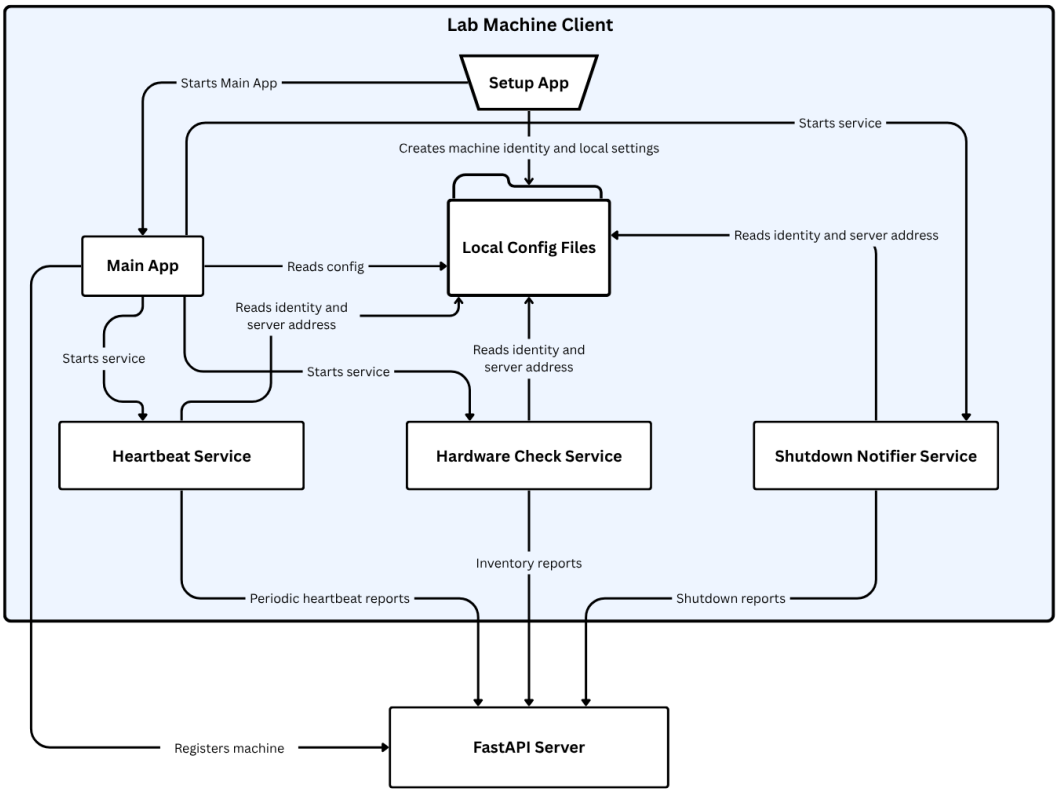

Figure 4.2: Client architecture showing the different client modules and how they communicate with each other

4.3.1 Client Installation and Initialization

A graphical application is provided to initialize the client software on each workstation. Its primary function is to generate a configuration file containing machine-specific information, including the machine's globally unique identifier (GUID), MAC address, room and computer name. The GUID is generated using Microsoft's GUID generation function [20] during the installation and initialization process of the client software.

This configuration file serves as a persistent local reference and is subsequently accessed by the service modules during runtime to fetch the machine information and server IP address. This file is read-only, which means that administrator privileges are required to modify it [21].

Once the configuration file has been created successfully, the application invokes an executable that runs a series of commands with administrator privileges. These commands install and start client services and initiate the registration handshake with the server. After the initial setup has been completed, the client services start automatically on each subsequent boot. No further manual intervention is required for the client software, unless the client software is uninstalled from the machine.

The client machines should also support Wake-on-LAN. Reliable Wake-on-LAN re-

quires configuration at both firmware and operating-system level on the test machines. Wake-on-LAN was enabled in BIOS/UEFI, deep sleep states were disabled, Windows Fast Startup was turned off, Modern Standby was disabled, and the network adapter was configured to accept magic-packet wake events through Device Manager.

4.3.2 Client Service Design

Each component of the client is implemented as an independent program and deployed as a Windows service executable. This design improves fault isolation, allowing other modules to continue to operate even if one module fails. Although such failures are not expected during normal operation, this follows good design practice by limiting the impact of unexpected faults and improving system robustness.

Implementing these processes as Windows services provides tight integration with the operating system and ensures that they can run continuously in the background independently of user sessions [19]. In particular, services can start automatically on system boot which means that heartbeat transmission and hardware inventory checks continue to run even when no user is logged in to Windows, making them suitable for monitoring shared workstations. This is important in a laboratory environment where machines may be idle, in use, or unattended at different times.

All client-side events are logged and retained, encompassing both routine operations and error conditions. Each module maintains a dedicated log file, ensuring clear separation of concerns and facilitating targeted diagnostics. Log files are stored within a designated directory ("clientDocs"), which serves as the central repository for the client's operational data. This directory also contains other essential configuration and state files, including the server address and the current hardware inventory profile. Logs are retained for seven days, giving operators time to investigate issues while preventing uncontrolled growth in storage usage.

4.3.3 Heartbeat Module

The heartbeat module is responsible for transmitting periodic status signals to the server at 30-second intervals. Each heartbeat consists of a minimal JSON payload that contains the GUID, uniquely representing the originating system. Communication is performed at the application layer using HTTP requests, while relying on TCP as the underlying transport protocol to ensure a reliable, ordered packet delivery under normal network conditions [22].

In addition to its primary implementation as a Windows service, the heartbeat mechanism is supplemented by a scheduled task designed to operate during sleep mode. The operating system's Task Scheduler invokes a lightweight executable (Heartbeat-once.exe), which is configured to wake the machine at fixed 30-second intervals. Upon execution, the task transmits a single heartbeat message, and then allows the machine to return to sleep [23].

The systems used for the implementation and evaluation of this project employed Modern Standby rather than the legacy sleep states commonly found in older systems. During testing, Modern Standby prevented the machines from being reliably awakened through both Task Scheduler wake timers and Wake-on-LAN mechanisms, so it was disabled on the test machines to make WoL-based recovery possible. As a result, the supplementary heartbeat service for low-power states was not needed in this implementation. However, it remains relevant for older systems that use traditional sleep states, since these can normally support scheduled wake events [23] and Wake-on-LAN without additional operating system configuration, making it unnecessary to disable sleep mode entirely.

The selection of a 30-second interval reflects a design trade-off between detection granularity and system flexibility. Shorter intervals enable finer control over server-side alert thresholds, allowing the system to tolerate a configurable number of missed heartbeats without introducing excessive delay in anomaly detection. In contrast, longer intervals would significantly increase the time required to confirm a failure condition. For example, with a five-minute interval, tolerating two missed heartbeats could delay detection by up to 15 minutes, creating a substantial window during which unauthorized hardware removal could occur without immediate detection. The 30-second interval also leaves time for Wake-on-LAN validation, which is further discussed in 4.5.

4.3.4 Hardware Inventory Module

The hardware inventory module is responsible for establishing the baseline profile during the initialization phase of the client. The baseline profile is used to create a local JSON file to store the machine's hardware configuration. This file is read-only and can only be edited by an administrator. The same information is transmitted to the server as part of the registration process. The hardware inventory module is also responsible for periodically retrieving the current hardware state at 30-second intervals, similar to the heartbeat frequency, and compares it against the local JSON file.

The system focuses particularly on GPUs, SSDs, and RAM, as these components are both critical for system functionality and commonly targeted in theft incidents. By prioritizing a limited set of high-value components, the system reduces unnecessary complexity while still capturing the most relevant events. Peripherals such as keyboards and mice are also included,. as they are more easily accessible and are more prone to removal or in shared environments. However, these components are generally considered less critical and may be subject to benign variations in usage, such as being temporarily moved or exchanged. Therefore, peripheral identifiers are not collected; the system only checks if the peripherals are connected.

To retrieve hardware information, the system uses Windows Management Instrumentation (WMI) [24]. This approach was chosen because WMI is a standardized Windows interface that provides structured access to system-level information without requiring external libraries or additional drivers. Through WMI, the client can

retrieve information about components such as RAM, storage devices, and GPUs in a structured and standardized way. This simplifies the implementation and ensures compatibility with the Windows-based workstations used in the project.

Similarly to the heartbeat module, this module is also implemented as a Windows service, allowing it to start automatically during system boot [19]. Upon launch, it verifies the existence of the local components file. If it does not exist, a new baseline is generated and transmitted to the server. During runtime, the module continuously compares the current hardware configuration against the latest-state file. When a change is identified, the updated configuration is sent to the server and the local file is updated. Similar to the heartbeat mechanism, this function does not operate while the system is in sleep mode. However, this limitation does not compromise system integrity, as discussed in Section 6.2.

4.3.5 Shutdown Notification Module

This module is responsible for sending a shutdown notification to the server to indicate that the machine is shutting down normally. During a controlled shutdown or reboot event, the module transmits a pre-termination notification to the central server before the machine shuts down.

This mechanism extends the standard shutdown procedure by introducing a communication step that explicitly signals the intentional termination of the machine. Such events may include routine system updates, scheduled restarts, or other administratively initiated actions. If a machine is shut down normally by a user, it also sends the notification. This gives the server additional context when later distinguishing expected shutdowns from abnormal loss of contact.

4.4 Server Architecture

The server is responsible for the operational state and the interpretation logic of the system. It maintains the registry of monitored machines, processes incoming heartbeat, inventory, and shutdown reports, and updates the latest known status of each workstation.

Reported hardware and peripheral information is evaluated against the latest known state and the approved baseline. Detected deviations are stored as structured machine change events, which can later be correlated and classified into alerts. In addition to machine-level analysis, the server can identify room-level patterns affecting multiple workstations within the same environment. The server also exposes API endpoints that support operator workflows such as listing, acknowledging, resolving, and dismissing alerts.

Figure 4.3 presents the main server-side components and their relationship to the rest of the system. It clarifies how incoming client reports pass through the API layer, how shared logic and alert processing operate on stored data, and how the

server supports both operator access and Wake-on-LAN recovery.

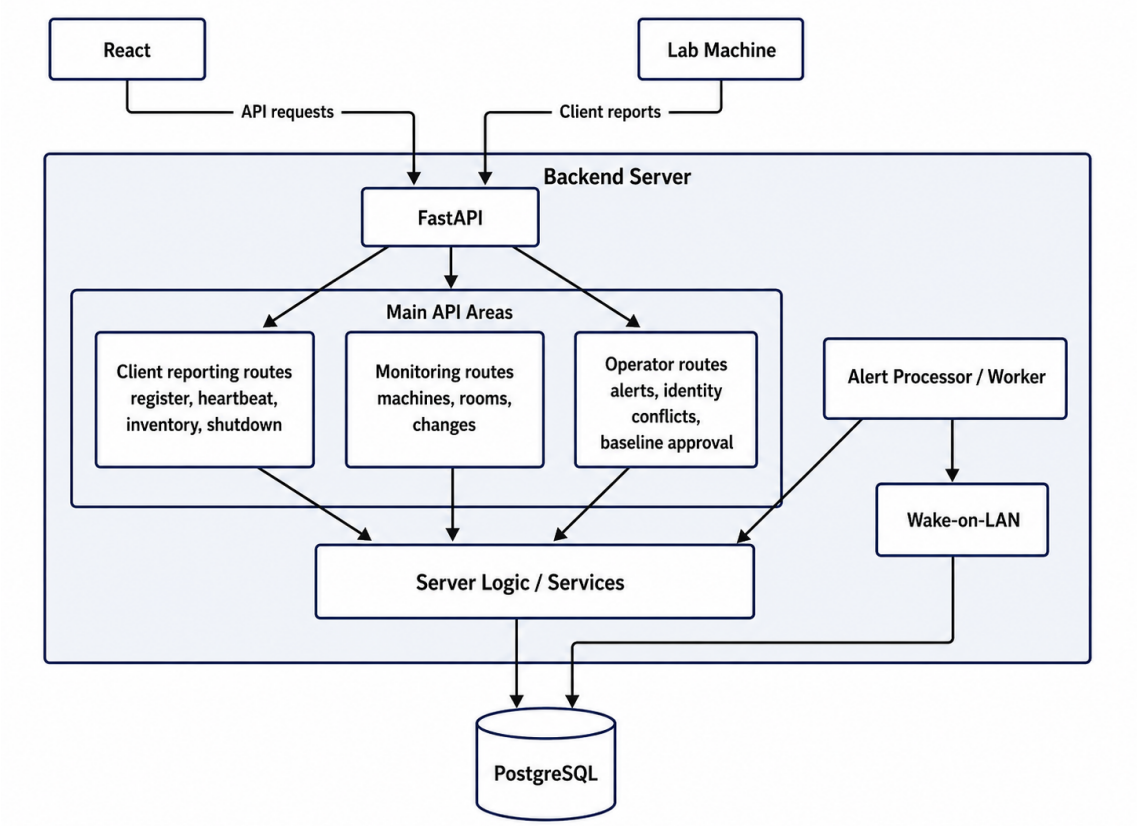

Figure 4.3: Server-side architecture showing the FastAPI API, main route groups, shared server logic, alert-processing worker, database interaction, and Wake-on-LAN support.

4.4.1 Machine Registry

Each physical workstation is represented by a main machine record containing its current machine GUID, room, computer name, first seen timestamp, and last seen timestamp. The registry also stores hardware and peripheral baseline definitions, as well as the latest reported inventory state for each workstation.

In addition to the main machine record, the server stores a mapping between the reported machine GUIDs and the main machine record. The MAC address is stored as a machine fingerprint and is used to recognize the same physical workstation. This is necessary because a machine GUID may change over time, for example, after re-installation or reconfiguration. The mapping table records whether a GUID is currently active or historical.

If a new machine GUID is reported with a MAC address that is already associated with an existing machine record, the server does not automatically overwrite the identity. Instead, it creates an identity conflict record that must be approved or rejected by an operator. This prevents accidental merging of machines and gives technicians control over identity changes.

Approving a conflict means that the newly reported GUID is accepted as belonging to the same physical workstation, based on the matching MAC address. The new GUID is then marked as the current identifier, while the previous GUID is stored as historical. Rejecting a conflict means that the reported GUID is not accepted as belonging to the existing workstation. In this case, the system does not automatically merge identities, and the new machine registration is ignored, giving operators control over whether the new identity should be accepted into the registry.

4.4.2 Database Schema

The database schema is centered on a main machine record, which represents one physical workstation in the system. This record stores the machine's current accepted GUID, room, computer name, and timestamps for when it was first and last seen. Data that changes during operation, such as inventory reports, heartbeat updates, detected changes, and alerts, is stored in separate related tables.

This separation allows the server to update the current machine state without losing the historical record needed for later investigation. Machine changes are therefore stored independently from alerts, so that alert generation can be treated as an interpretation of recorded evidence rather than as the only record of an event. The complete database schema is provided in Appendix B.

4.4.3 Inventory and Change Detection

When a machine is registered for the first time, the server stores the reported hardware and peripheral state as both the initial approved baseline and latest known state. The approved baseline remains unchanged unless explicitly updated by an operator, while the latest state is continuously updated based on incoming inventory reports.

This comparison produces structured change events. For example, if the reported GPU differs from the approved baseline, the server records a GPU baseline deviation event. Similar checks are performed for RAM, SSD, keyboard, and mouse state.

The system distinguishes between changes relative to the latest known state and deviations from the approved baseline. This reduces duplicate events because a persistent deviation is not repeatedly inserted every time the client sends the same inventory report.

4.4.4 Heartbeat and Availability State

When a heartbeat message is received, the server validates that the reported machine GUID is registered as the current GUID for a known workstation. If the identifier is valid, the server updates the machine's last seen timestamp.

Availability is derived from this timestamp. A machine is considered online while its last seen timestamp is within the configured offline threshold, and offline when

that threshold is exceeded. This state is used by the alert processor as input for later evaluation.

4.4.5 Shutdown Reporting

The shutdown endpoint allows a client to report that a machine is shutting down. The server validates the reported machine GUID, updates the machine's last seen timestamp, and stores the shutdown as a machine change event. Recording shutdowns as machine change events gives the alert processor context for later interpretation, acting as potential evidence of a benign termination as discussed in Section 4.3.5.

4.5 Wake-on-LAN Recovery and Validation

Wake-on-LAN is used as a recovery and validation mechanism when a workstation stops reporting. After a missed heartbeat interval, the alert processor can send a magic packet to the MAC address stored for that machine record. The mechanism repeats this recovery attempt while the machine remains unavailable, so that temporary low-power states can be handled without requiring manual intervention.

Each recovery attempt sends three magic packets with a one-second interval between packets. Repeating the packet improves the likelihood that the wake request reaches the network adapter, while the short spacing avoids introducing unnecessary delay before the alert processor continues evaluating the machine state. If the machine resumes operation, its next heartbeat updates the server's last seen timestamp. If it does not resume, the machine remains unavailable and can be evaluated by the alert logic at a later stage.

This mechanism helps distinguish recoverable low-power or shutdown-related states from cases that require further attention. It does not by itself decide whether an incident has occurred, but provides additional evidence before alerts are classified or escalated.

4.6 Context-Aware Alarm Logic

The server uses rule-based alert logic because alerts must be interpretable and traceable to specific system conditions. The alert processor evaluates multiple sources of information: recorded machine change events, availability state derived from heartbeat timestamps, and recovery results from Wake-on-LAN attempts. It also takes into account machine identity status, room assignment, and laboratory opening hours when deciding whether an alert should be created or escalated.

The system separates alert classification into several fields. The alert type describes what condition was detected, such as a hardware deviation or a room-level offline pattern. The severity describes the operational importance of the alert, consisting

of warning, high, or critical. The workflow status describes how the alert has been handled by an operator, such as open, acknowledged, resolved, or dismissed.

Hardware deviations involving GPU, RAM, or SSD are treated as serious because these components are relevant to physical tampering or theft. When such a deviation is detected, the server normally creates a critical machine-level alert. If the affected machine has a pending identity conflict, the alert is instead downgraded to warning severity and is not automatically escalated to critical. This reduces false escalation in cases where legitimate maintenance or re-installation temporarily creates uncertainty about machine identity. The related security trade-off is discussed further in Section 6.2.

Room-level alerts are used for patterns that affect several machines in the same room. Shutdown spike detection is based on recent shutdown reports followed by failed Wake-on-LAN recovery and is marked as critical when the rule condition is met. Offline spike detection is based on multiple machines exceeding the offline threshold during configured laboratory opening hours; these alerts are created with high severity and escalate to critical only when the configured offline escalation condition is satisfied. The exact rule thresholds are specified in Section 4.6.1.

Figure 4.4 summarizes how the alert processor combines incoming reports, stored state, and recovery information before creating or suppressing alerts. The simplified view shows the main decision path, while the full processing sequence is included in Appendix A.

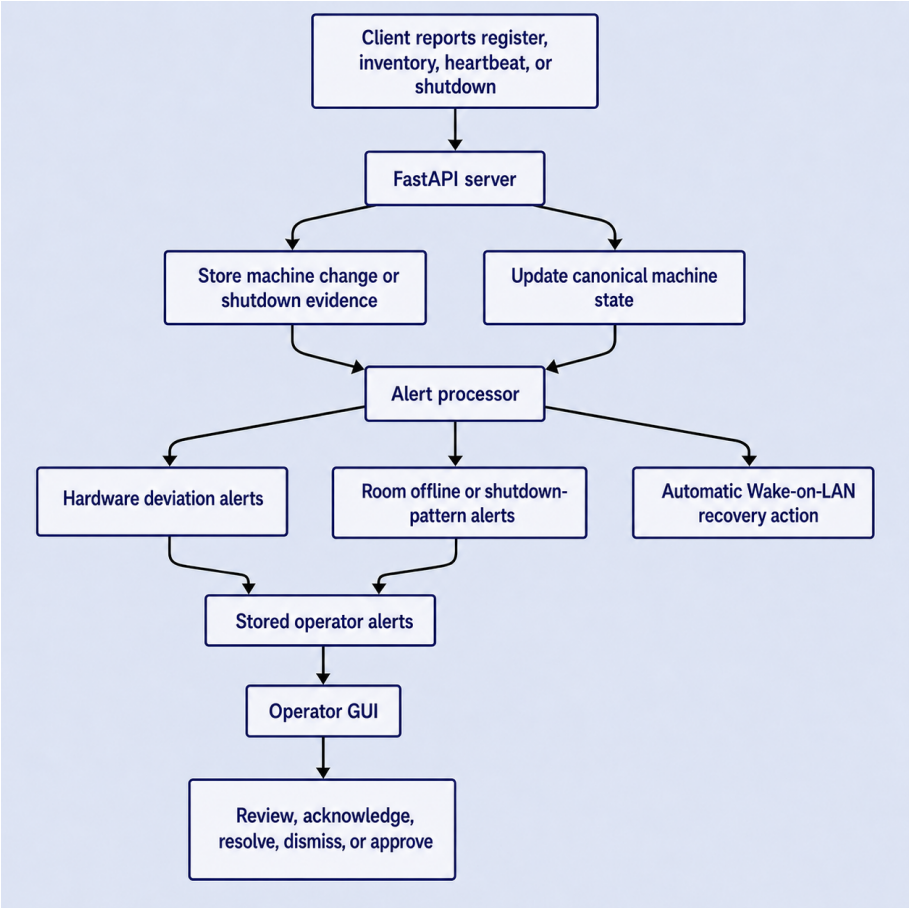

Figure 4.4: Simplified flow of the context-aware alert logic, from incoming machine reports and stored state to alert generation and recovery actions.

4.6.1 Implemented Alert Rules

In order to distinguish suspicious incidents from benign incidents, the server applies a set of rule-based conditions. Hardware baseline deviations involving GPU, RAM, or SSD generate critical machine-level alerts. As discussed in Section 4.3.4, peripheral state changes are more susceptible to benign variation in shared environments. Therefore, keyboard and mouse changes are recorded as machine changes, but do not generate alerts.

At room level, the server creates a shutdown spike alert when at least three machines in the same room report shutdowns within a ten-minute window and do not recover after the automatic recovery process. A room offline spike alert is created when at least three machines in the same room exceed the one-minute offline threshold during configured laboratory opening hours, which in the prototype are weekdays from 08:00 to 18:00. These thresholds were selected to avoid alerts for isolated workstation failures while still detecting correlated room-level incidents.

Hardware baseline deviations and room shutdown spikes are critical by default. If a hardware deviation affects a machine with a pending identity conflict, however, the alert is downgraded to warning severity and is not automatically escalated. Room

offline spikes are likewise escalated when they are created, meaning that correlated room-level availability failures are treated as potential security incidents rather than solely operational faults.

Room-level shutdown and offline alerts are automatically resolved when the corresponding room condition is no longer active. The alert processor reevaluates these conditions every 15 seconds, allowing alerts to be updated shortly after new evidence arrives, while keeping alert evaluation separate from individual client requests.

4.6.2 False Alarm Mitigation

Several mechanisms are used to reduce false alarms. Isolated shutdowns and isolated offline machines do not automatically create alerts. Instead, the system focuses on correlated patterns across multiple machines in the same room. Repeated room-level alerts of the same type are suppressed for 30 minutes, giving operators time to notice and investigate an incident without allowing the same condition to generate multiple notifications. The alert processor also de-duplicates machine-level hardware alerts by updating an existing open or acknowledged alert with new evidence instead of creating a separate alert.

4.7 Reporting and Notification Design

The implemented server exposes alerts and machine status information through API endpoints. These endpoints provide structured access to the system state and are used by the graphical user interface and can be used by potential external notification services.

4.7.1 Alert Status Workflow

Alerts can have different statuses depending on how operators handle them. An open alert is newly generated and has not yet been reviewed. Acknowledging an alert marks it as seen by an operator, but does not close it. Resolving an alert means that the underlying issue has been investigated and addressed, so the alert can be closed. Dismissing an alert means that the alert is considered non-relevant, incorrect, or caused by a benign event, and therefore requires no further action.

4.7.2 Real-Time Notifications

In the current prototype, notifications are presented to operators through the graphical user interface. Alerts are stored in the database and made available via API endpoints, allowing them to be displayed in the interface as they are retrieved from the server. This enables operators to monitor the state of the system and respond to emerging issues.

An important design principle is that notifications are framed as decision support rather than definitive conclusions. The system does not assert that theft or failure

has occurred, but instead indicates that a situation may require further investigation by an operator.

4.8 Graphical User Interface

The system provides a graphical user interface that serves as the primary point of interaction for personnel. The interface is organized into several complementary views, supporting aspects of monitoring, investigation, and system maintenance. A sidebar is displayed persistently on the left side of the interface and provides direct access to the main views of the system, `Dashboard`, `Rooms`, `Machines`, `Alerts`, `Changes`, and `Identity Conflicts`.

4.8.1 Dashboard View

The dashboard view, shown in Figure 4.5, presents a condensed overview of the current system state. Summary cards at the top display key metrics, including the total number of registered machines, the number currently online or offline, the number of open alerts, and the number of pending identity conflicts. Beneath these cards, a panel lists recent open alerts together with their titles, affected room or machine, severity level, and timestamp. A link to the full `Alerts` page allows the user to navigate from this overview to more detailed investigation.

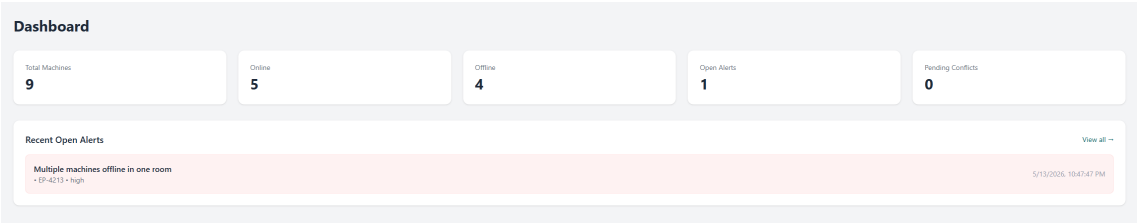

Figure 4.5: Dashboard view presenting summary metrics and recent open alerts.

4.8.2 Rooms View

The rooms view, shown in Figure 4.6, provides an aggregated overview of the monitored environment at the room level. Each room is represented by a summary card displaying the total number of machines, the number currently online or offline, and the availability of connected peripherals. A room status indicator further highlights whether the room is operating normally or requires attention.

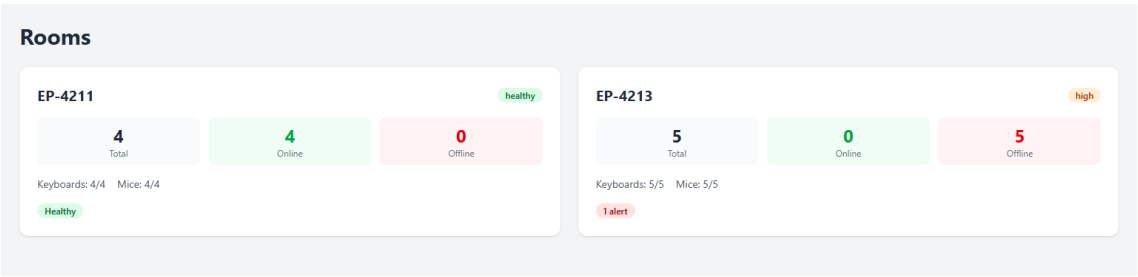

Figure 4.6: Rooms view displaying aggregated status information for each monitored room.

Selecting a room opens a more detailed room view, shown in Figure 4.7. In addition to the same room-level summary metrics, this page presents a table of the machines registered in the selected room, including their machine identifiers, PC names, current online or status, and most recent contact time. Each entry also provides a link to the corresponding machine detail view. The room detail view further includes room-scoped information about active alerts and pending identity conflicts.

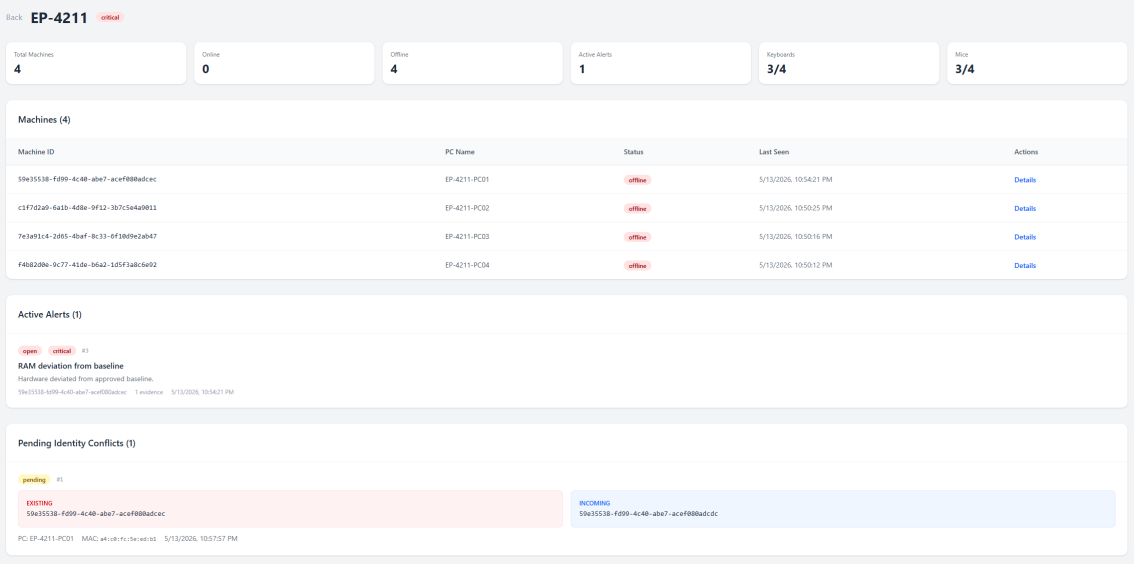

Figure 4.7: Room detail view showing room-level metrics, machine listings, active alerts, and pending identity conflicts.

4.8.3 Machines View

The machines view, shown in Figure 4.8, provides a complete list of all monitored machines. For each machine, the table displays its machine identifier, PC name, assigned room, current online or offline status, and most recent contact time. To support efficient lookup, the view includes a search field for filtering by machine identifier, room, or PC name, together with controls for restricting the list according to status. Each row also contains a link to a dedicated machine detail view.

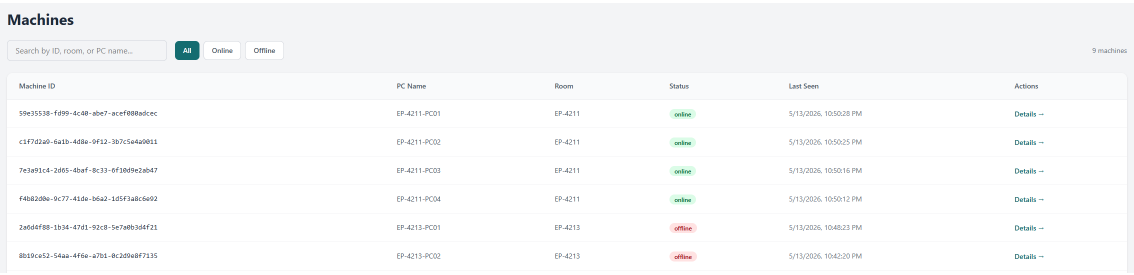

Figure 4.8: Machines view displaying all monitored machines with online or offline status, location, and filtering options.

Selecting the Details button opens the detailed machine view shown in Figure 4.9. This view presents identity and timing information, including the PC name, machine

identifier, assigned room, current status, and first and last seen timestamps. It also displays the approved baseline inventory alongside the most recently reported inventory, enabling direct comparison of hardware and peripheral state. In particular, operators can inspect deviations in components such as the GPU, RAM modules, total RAM capacity, SSD, keyboard, and mouse. When a hardware change is legitimate, the latest reported configuration may be approved as the new baseline.

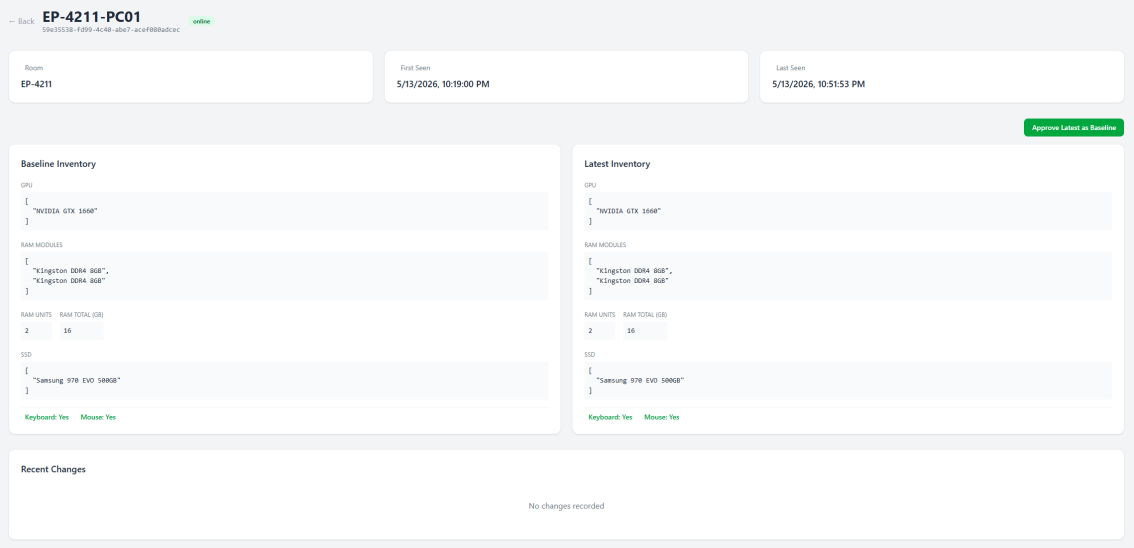

Figure 4.9: Machine detail view showing identity information, comparison between baseline and latest reported inventory, and recent changes.

4.8.4 Alerts View

The alerts view, shown in Figure 4.10, provides a dedicated interface for reviewing and managing alerts generated by the monitoring system. Alerts can be filtered using the `All`, `Open`, `Acknowledged`, `Resolved`, and `Dismissed` controls, each of which displays the current number of alerts in that category. For each alert, the view presents its status, severity level, title, descriptive summary, affected machine or room, and timestamp. Action buttons allow operators to acknowledge, resolve, or dismiss alerts as more information becomes available.

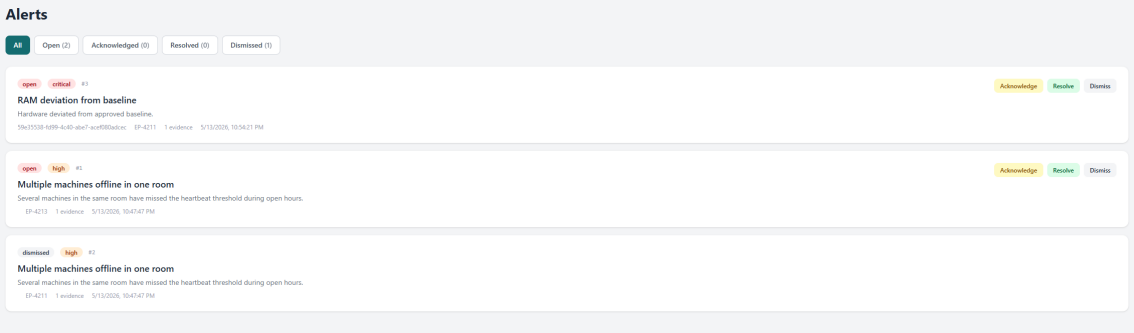

Figure 4.10: Alerts view showing generated alerts with workflow-state filters and available response actions.

4.8.5 Changes View

The changes view, shown in Figure 4.11, provides a chronological record of detected changes across the monitored environment. Each entry includes the affected machine, the type of change, a structured JSON payload containing the reported change data, and the corresponding timestamp. To support targeted inspection, the view includes a search field for filtering by machine identifier or change type, together with a dropdown menu for selecting specific categories of changes. This view is intended to support more detailed analysis by enabling operators to trace historical events and inspect the data associated with each recorded change.

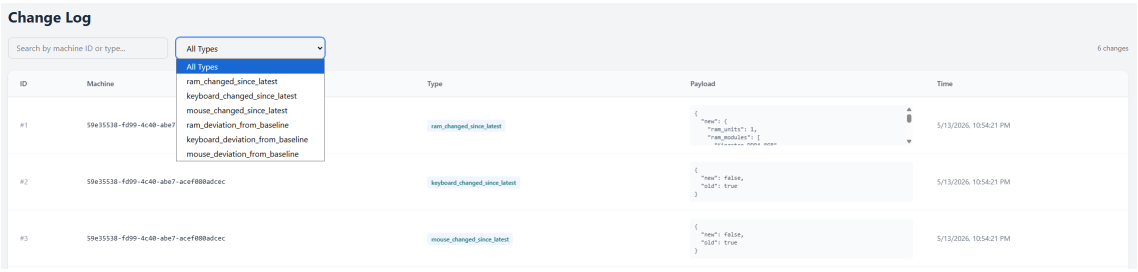

Figure 4.11: Changes view presenting a log of detected changes with search and type-based filtering options.

4.8.6 Identity Conflicts View

The identity conflicts view, shown in Figure 4.12, provides a dedicated interface for resolving cases in which an incoming machine identity conflicts with an existing registered identity. Conflicts can be filtered by workflow state using the `All`, `Pending`, `Approved`, and `Rejected` controls. Each record presents a sequential identifier together with a side-by-side comparison of the existing and incoming machine identifiers. Additional contextual information, including room, PC name, MAC address, and detection time, is also displayed to support informed review. Operators can then approve or reject each conflict using the available action controls.

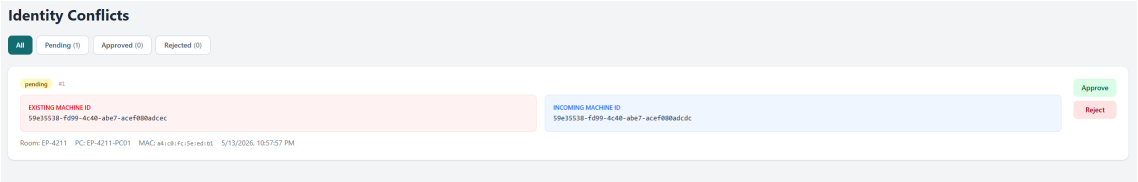

Figure 4.12: Identity conflicts view showing detected identity conflicts and available resolution actions.

5

Evaluation

This chapter evaluates the system against a set of predefined test scenarios covering both normal operation and abnormal conditions representative of potential security incidents. The goal is to assess whether the system correctly detects relevant events, tolerates disturbances without generating false alarms, and appropriately escalates failures.

The evaluation used the threshold values and the escalation rules defined in Section 4.6.1. In the test configuration, clients transmitted heartbeat messages every 30 seconds, and a machine was classified as offline after exceeding the one-minute offline threshold. Because heartbeat transmission and alert evaluation occur periodically, detections based on the one-minute offline threshold may be observed slightly after the threshold is exceeded. Room-level offline alerts were generated when at least three machines in the same room exceeded this threshold during configured laboratory opening hours. Room-level shutdown alerts were generated when at least three machines in the same room reported shutdowns within a ten-minute window and did not recover after the automatic recovery process. The alert processor reevaluated these conditions every 15 seconds.

5.1 Test Environment

The evaluation was conducted in a physical computer laboratory at Chalmers University of Technology containing four workstations connected to the campus network via Ethernet through the same switch. The machines remained powered on continuously during the testing. To perform the tests, one machine was designated as the server host. The remaining three machines ran the client software and served as test targets.

5.2 Test Scenarios

Most scenarios were executed five times on three different machines to account for variability in system behavior. Scenarios involving internal hardware removal were conducted five times on a single machine.

5.2.1 Normal Operations

After installation, all three clients successfully generated their initialization files and transmitted registration requests to the server. Each request contained a JSON payload describing the machine's hardware inventory and connected peripherals. The system was then left to run without intervention for one hour.

Throughout this period, all clients continued transmitting heartbeat signals at the configured 30 second interval, including when no user was logged in, confirming that the service runs independently of user sessions. The server received all heartbeat messages and raised no alerts, establishing a baseline with no false alerts.

At the end of the test, the machines were shut down using the operating system's standard shutdown function. Shutdown notifications were successfully transmitted and received in all cases.

5.2.2 Network Interruption

Network connectivity was interrupted for durations both shorter than and exceeding the configured offline threshold. During the shorter outage, one or more heartbeat messages failed to reach the server. Once connectivity was restored, the clients resumed transmitting heartbeats and the server updated the last-seen timestamps accordingly. Because the brief interruption did not exceed the offline threshold, no machine was classified as offline and no alert was generated. This confirms that the system tolerates temporary network disturbances without misclassifications.

During the prolonged interruption, the server stopped receiving heartbeats from the affected machine within the expected time window and subsequently classified it as offline. When a single machine was affected, the event was recorded but not escalated as a room-level alert, consistent with the conservative handling of isolated failures. When the same condition was applied to all three machines in the same room simultaneously, the server identified it as a correlated outage and generated a room-level offline alert, within 90 seconds.

5.2.3 Power Cable Disconnection

The power cable was disconnected from one client machine, causing an abrupt power loss. The machine immediately stopped transmitting heartbeats, and no shutdown notification was sent. Within 90 seconds, the server classified the machine as offline. Because only one machine was affected, no room-level alert was generated.

The power cables were then disconnected from all three client machines in rapid succession. All affected machines stopped transmitting heartbeats at approximately the same time and, again, sent no shutdown notifications. Within 90 seconds, the server detected that three machines in the same room had become unavailable within the same time window and generated a room-level offline alert.

5.2.4 Wake-on-LAN Recovery

A machine was turned off using the operating system's standard shutdown function. It transmitted a shutdown notification to the server and entered a low-power state. After one heartbeat interval (30 seconds), the server started transmitting Wake-on-LAN packets to the machine. The machine successfully resumed operation and began transmitting heartbeats again. No hardware discrepancy was reported, so the event was not escalated further.

A machine's power cable was then unplugged and after one missed heartbeat, the server started transmitting WoL packets. The machine did not resume operation and no heartbeat was received since it was unplugged. When the server did not receive the second missing heartbeat, it classified the machine as offline. This occurred within 90 seconds.

5.2.5 Internal Hardware Removal

A machine was shut down by unplugging the power cable. One of the machine's two RAM modules was then physically removed, reducing the RAM configuration from two modules to one. After 3 minutes from the shut down, the power cable was plugged in and within 30 seconds the computer was powered on by Wake-on-Lan. Upon startup, the client resumed normal operation by transmitting a heartbeat signal, followed by a hardware inventory update indicating the changed RAM configuration within 30 seconds after the system booted. The same procedure was then repeated separately with the removal of the GPU. In both cases, the system correctly identified the hardware discrepancy by comparing the new inventory against the stored baseline and reported the change to the server. An alarm was generated within four minutes of the physical intervention.

5.2.6 Peripherals Removal

The hardware inventory module periodically polls for connected peripherals at 30-second intervals. During this test, the keyboard and mouse were deliberately disconnected from the client machine. The client detected the disconnections and transmitted the corresponding change notifications to the server within 30 seconds, consistent with the polling interval.

5.3 Result Analysis

The system successfully detected both availability-related events and hardware changes. The loss of heartbeat messages beyond the configured threshold resulted in correct classification of machines as offline. Hardware inventory monitoring detected both internal component changes and external peripheral disconnections.

5.3.1 False Alarm Resistance and Event Correlation

No false alarms were generated during normal operation or brief network interruptions. This indicates that the system tolerates temporary disturbances without misclassification. The Wake-on-LAN mechanism contributed to reducing unnecessary alerts by verifying whether a machine could recover from a low-power state before escalation.

The system also distinguished between isolated failures and correlated events. Single-machine outages were handled conservatively, while simultaneous outages affecting multiple machines resulted in room-level alerts. This behavior supports the intended design of reducing unnecessary escalation while still identifying important incidents. At the same time, this implies that availability-only incidents affecting a single machine may not trigger high-priority room-level alerts unless accompanied by a detected hardware baseline deviation.

5.3.2 Latency and Alarm Timing

An important aspect of the system is timely detection of events. For scenarios that resulted in offline classification or room-level alerts, the system reacted within 90 seconds. Hardware removal detection, however, depends on how quickly the computer is reconnected to power after tampering, since the system can only observe the missing component once the machine is back online. In the internal hardware removal test, where the computer was promptly reconnected, the alarm was generated within four minutes. If the computer remains offline for an extended period, detection is delayed accordingly. Furthermore, if the removed component is essential for booting, the hardware removal would go undetected because the computer cannot be turned on. Still, operators would be able to observe the machine as permanently offline within 90 seconds.

5.3.3 Summary

The prototype demonstrated the correct behavior in all tested scenarios. It successfully registered clients, maintained availability monitoring, detected hardware and peripheral changes, and differentiated between isolated and correlated incidents. The evaluation provides confidence that the system meets its primary functional requirements under normal operating conditions.

6

Discussion

The work presented demonstrates that the cause of computer unavailability can be interpreted to a meaningful degree by combining a small number of complementary signals. The key insight is that no single signal is sufficient on its own. A heartbeat timeout alone tells the server only that a machine is no longer reachable; it does not explain why. By layering contextual checks, the system constructs a richer picture that allows it to distinguish between different types of events and reduces the false-alarm rate.

At the same time, the evaluation revealed limitations of what can be achieved with the signals that were available within the project scope. Events that mimic legitimate behavior cannot be distinguished from genuine benign events without additional context from outside of the system, such as room occupancy sensors. The research question mentioned in Section 1.3 is therefore answered with a condition: the available system and network context is sufficient to handle many common cases, but a fully robust solution would benefit from more environmental data.

6.1 Rule-Based Logic versus Machine Learning

A deliberate design decision in this project was to implement the alarm logic as a set of hand-crafted rules rather than as a trained machine learning model. This choice was motivated primarily by the need for interpretability and the limited project scope.

In practice, the rule-based approach proved effective for the scenarios tested. Its principal strength is transparency: every alarm or suppression of an alarm can be tracked to the specific rules and the data values that triggered it. For an operational system where operators must trust and understand the alerts they receive, this traceability is a significant advantage.

However, the approach also has limitations. Defining appropriate thresholds required iterative manual calibration, and the values that worked well in the test environment are not guaranteed to generalize. A set of thresholds tuned for one computer room may produce too many or too few alarms in a room with a different usage pattern. In addition, adding new contextual rules is straightforward in isola-

tion, but increases the overall complexity of the rule set, making it harder to reason about interactions between rules over time.

A machine learning approach could, in principle, learn environment-specific patterns from data and adapt to new settings with less manual effort. Whether this is practical largely depends on the availability of sufficient labeled data. The applicability of such an approach is further discussed in Section 6.4.

6.2 Limitations

Several limitations of the current work should be acknowledged:

1. **Limited evaluation scope:** The system was mainly evaluated in a controlled setting with a small number of machines over a short period. This is sufficient to validate the core detection mechanism but does not capture the full range of conditions that a production deployment could encounter. Although the architecture was designed to support many monitored workstations through lightweight clients, centralized processing, and configurable thresholds, the prototype was not load-tested at scale. Therefore, the evaluation cannot demonstrate how the server, database, network traffic, or operator interface would behave with hundreds or thousands of machines.

2. **Environment-specific calibration:** The thresholds and weighting factors used in the rule-based logic were calibrated for the specific test environment. Deploying the system in a different room or building would likely require recalibration, and there is currently no automated procedure to do so.

3. **Hardware detection limitations:** Hardware inventory checks cannot be performed while the machine is in sleep mode, which may limit detection. Although heartbeat signals continue to be transmitted through the Task Scheduler mechanism, allowing the server to maintain awareness of the machine's online status during low-power states, direct hardware validation remains inactive during this period. From a threat-model perspective, however, this limitation is not considered critical. In realistic theft scenarios, hardware removal is unlikely to occur while the system is in sleep mode. A malicious actor would typically either wake the machine in order to perform a controlled shutdown before removing components, or forcibly disconnect the power supply. In both situations, the machine exits the sleep state, resulting either in the transmission of a shutdown notification or in the loss of heartbeat signals. Consequently, the absence of hardware monitoring during sleep mode does not create a substantial detection gap, since any event involving physical component removal inherently requires a state transition that is already observable through the existing monitoring mechanisms.

4. **Wake on LAN network limitations:** The current WoL implementation assumes that wake requests can reach the target workstation on the local network. In practice, WoL packets are typically limited to the same subnet

unless the network infrastructure is configured to forward them. This may limit scalability in a larger deployment across multiple rooms, buildings, or network segments.

5. **Identity conflict interpretation:** When a reported GUID does not match the previously known identity of a physical workstation, the prototype flags this as an identity conflict. An operator must then approve or reject the discrepancy before the identity change is accepted by the system. This is useful in legitimate administrative situations, such as reinstallation, service work, or approved hardware replacement. However, hardware-related alerts associated with pending identity conflicts are assigned warning severity rather than critical severity, which introduces a security trade-off. A malicious actor could theoretically attempt to trigger an identity conflict in order to reduce the apparent severity of a hardware deviation. For this reason, identity conflicts should not be interpreted as benign automatically, and any hardware deviation occurring during an unresolved identity conflict should be reviewed carefully by an operator.

6.3 Societal and Ethical Aspects

This project involves monitoring computer workstations that are used by students in shared computer rooms. Although the system is primarily technical in nature, it raises several societal and ethical considerations. Monitoring in educational environments may influence perceptions of privacy, trust and transparency [25]. Therefore, it is important to assess how the system may affect students and staff, even if no personal data is collected. The following subsections discuss key aspects related to privacy, responsible data usage and societal impact.

6.3.1 Privacy and Data Protection

The system is designed to monitor technical workstation behavior, such as hardware status and operational anomalies, rather than individual user activity. No personal data or information related to specific students is collected or processed. Consequently, the risk of direct privacy violations is considered limited.

However, even systems that do not process personal data can create a perceived sense of surveillance. Such perceptions can influence behavior and potentially reduce trust in the educational environment [26]. For this reason, privacy considerations extend beyond legal compliance and include the broader ethical responsibility to minimize unnecessary data collection.

To address these concerns, the system was designed to collect only information strictly necessary for operational monitoring. This design follows the principle of data minimization, which states that collected data shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed [27]. In addition, access to monitoring data is restricted to authorized personnel.

6.3.2 Ethical Use of Monitoring Data

Another ethical aspect concerns how IT and security personnel use monitoring data and alarms. There is a risk that poorly designed monitoring systems can lead to unnecessary investigations or misuse of information [28]. To mitigate this, the system is designed to distinguish between technical anomalies and suspicious incidents, and alarms are presented as decision support rather than definitive evidence of wrongdoing.

6.3.3 Societal Impact and Trust

From a societal perspective, the system aims to improve the availability and reliability of shared educational resources. Reducing theft and downtime contributes to a fairer and more efficient use of university infrastructure. At the same time, maintaining student trust is essential. If students are unaware of what the system monitors, or if they incorrectly assume that their personal activity is being tracked, the system may negatively affect the learning environment rather than improve it.

To mitigate this, transparency measures should accompany any deployment of the system. These include placing clear notices in monitored computer rooms stating the purpose of the system and clarifying that no personal data or user activity is collected; making a description of the system, its scope, and the data it collects available through the university's IT department website; and informing students through existing communication channels such as course platforms or student newsletters when the system is first introduced. These measures would help ensure that students understand the intent and limitations of the monitoring system, reducing the risk of misperception and supporting continued trust in the shared learning environment.

At the same time transparency should be balanced against security considerations. Public information about the system should explain its purpose, scope, and data handling, but should not disclose operational details such as exact alarm thresholds or detection rules. This reduces the risk that the information could be used to circumvent the system while still allowing students to understand how monitoring data is collected and used.

6.4 Future Work

Several directions remain for future development. Most importantly, the prototype should be evaluated over a longer period and across multiple computer rooms with varying hardware configurations and usage patterns. A sustained deployment would reveal whether the calibrated thresholds remain stable as usage patterns shift and would provide a sufficiently large dataset against which to assess alarm accuracy. Feedback from operators during such a trial would also highlight usability issues and missing features that are difficult to anticipate in a laboratory setting.

6.4.1 Support for Additional Operating Systems

The current system is limited to Windows-based workstations, as specified in the project scope. Extending the client to support the Linux operating system would broaden the system's applicability to environments where this operating system is used as well. The core architecture of the system is not inherently platform-specific, but the mechanisms for tasks like collecting hardware data and sending shutdown messages would need to be re-implemented using platform appropriate solutions.

6.4.2 Improved Hardware Identification

The current system relies on WMI for hardware detection, which may not always accurately reflect underlying physical components. In the current implementation, certain devices identified as virtual are filtered out in order to reduce irrelevant data. While this approach improves clarity and worked well in the tested environment, it may also introduce a risk of excluding relevant physical components in some configurations. Further investigation is therefore needed to evaluate the impact of this filtering and ensure that no critical hardware is omitted.

Additionally, further evaluation in a wider range of environments is necessary. For example, the current system has primarily been tested on machines equipped with a single SSD, which limits the diversity of observed configurations. Testing the system in environments with more complex setups, such as multiple storage devices or virtualized storage solutions, could help identify additional limitations and improve the overall robustness and reliability of the system.

6.4.3 Maintenance Mode and Identity Review Workflow

A future version of the system could include an explicit maintenance mode for workstations undergoing service, re-installation, or approved hardware replacement. This would allow technicians to mark a machine as temporarily under maintenance before changes occur, rather than relying on identity conflicts as an indirect indication of possible maintenance activity.

Such a workflow could reduce unnecessary critical alerts while preserving security. Expected changes could be handled with less urgency during an approved maintenance window, while unexpected identity conflicts or hardware deviations outside such a window would still require careful operator review.

6.4.4 Machine Learning Based Alarm Logic

As discussed in Section 6.1, the current rule-based alarm logic requires manual threshold calibration and may not generalize between environments. A possible extension would be to explore machine learning techniques for the classification step. For example, an anomaly-detection model could be trained on historical heartbeat patterns and contextual features to learn what "normal" looks like for a given room and flag deviations. Such an approach would require a labeled data set of benign

and suspicious events, which could be collected over time from the logs and operator feedback of the current system. In practice, this would mean that operators annotate each alert as a true positive or false positive, gradually building a training dataset. Any such approach would need to preserve the interpretability that is a strength of the rule-based approach.

6.4.5 Integration of Physical-Context Data

Although presence sensors were excluded from the scope of this project, incorporating physical context data such as room occupancy information and door access logs could significantly improve the classification accuracy. For example, a computer losing connection to the server while a room is unoccupied would be less suspicious than if someone were present in the room. Integrating such data sources would require careful consideration of privacy implications, since room-access and occupancy data may be linked to identifiable individuals.

6.4.6 Scalable Wake on LAN Support

Future versions of the system should investigate how WoL can be made scalable across multiple network segments. In the current prototype, WoL is mainly suitable for machines on the same subnet as the server or the WoL sender. For larger deployments, network switches or routers could be configured to support directed broadcast, relay wake requests, or otherwise forward WoL packets to the correct subnet.

Another possible approach would be to deploy local WoL agents per room or network segment. The central server could then request the local agent to send the WoL packet within the target machine's subnet. This would preserve the usefulness of WoL as a validation mechanism while making it more practical for a larger campus-wide deployment.

7

Conclusion

This project aimed to design and evaluate a monitoring and alert software that is capable of interpreting the cause of computer unavailability in shared computer laboratory environments.

The central research question concerned the detection of potentially suspicious physical interventions. Specifically, it investigated whether data communicated between a server and client software could be integrated with contextual information to differentiate such events from benign ones. A further constraint was that the approach had to maintain an acceptably low false alarm rate.

The prototype demonstrates that layering multiple complementary signals, including heartbeat monitoring, Wake-on-LAN, and hardware inventory comparison, enables meaningful interpretation of workstation unavailability. The alarm logic successfully suppressed alarms for common benign events, such as individual reboots and brief network interruptions, while detecting scenarios involving hardware removal and multiple shutdowns.

The main limitation is that some scenarios cannot be differentiated using system level data alone, as different events may produce the identical signatures. Extending the system with physical-context data, such as room occupancy information, and exploring machine-learning classification are the most promising directions for future work.

Overall, the project shows that a lightweight, interpretable, and cost-effective monitoring solution can meaningfully improve the ability to detect and respond to hardware-related incidents in shared computer environments.

Bibliography

[1] A. Linde, private communication, Feb. 2026.

[2] L. Hensler, "Over $10,000 of equipment stolen from Friend Center lab." The Daily Princetonian. Accessed: Nov. 5, 2024. [Online]. Available: https://www.dailyprincetonian.com/article/2024/11/princeton-news-broadfocus-equipment-stolen-friend-center-computer-science-gpus-lab-heist.

[3] Rahul Awati, "What is geofencing?" Techtarget. Accessed: May. 07, 2026. [Online]. Available: https://www.techtarget.com/whatis/definition/geofencing

[4] "Prey Full Suite," Prey. Accessed: Feb. 10, 2026. [Online]. Available: https://help.preyproject.com/article/185-overview-what-is-prey.

[5] "What is lansweeper?" Lansweeper. Accessed: Apr. 28, 2026. [Online]. Available: https://docs.lansweeper.com/docs/what-is-lansweeper.

[6] "N-central architecture guide," N-able User Guide. Accessed: Mar. 31, 2026. [Online]. Available: https://documentation.n-able.com/N-central/userguide/Content/Further_Reading/Architecture_Security/architecture_guide.htm.

[7] Z. Hou, Y. Huang, S. Zheng, X. Dong, and B. Wang, "Design and implementation of heartbeat in multi-machine environment," in *Proceedings of the 17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003*, Xi'an, China, 2003, pp. 583–586. [Online]. Available: https://www.computer.org/csdl/proceedings-article/aina/2003/19060583/12OmNwGZNJL

[8] P. Matczak, A. Wójtowicz, A. Dąbrowski *et al.*, "Cost-effectiveness of cctv surveillance systems: Evidence from a polish city," *European Journal on Criminal Policy and Research*, vol. 29, no. 4, pp. 555–577, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s10610-022-09527-5

[9] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," Accessed: Apr. 25, 2026. [Online].

Available: https://agilemanifesto.org/.

[10] I. Sommerville, *Software Engineering*, 8th ed. Boston, MA, USA: Pearson Addison Wesley, 2006.

[11] S. Mbuguah, V. Mony, and G. Nyabuto, "Client-server architecture, a review," *International Journal of Scientific Research & Engineering Trends*, vol. 3, no. 2, Jan. 2024. [Online]. Available: https://ijasca.org/index.php/ijasca/article/view/48

[12] "Go documentation," Go. Accessed: Apr. 28, 2026. [Online]. Available: https://go.dev/doc/

[13] "Fastapi," FastAPI. Accessed: Apr. 28, 2026. [Online]. Available: https://fastapi.tiangolo.com/.

[14] "Postgresql documentation," postgresql. Accessed: Apr. 28, 2026. [Online]. Available: https://www.postgresql.org/docs/

[15] "Transactions," postgresql. Accessed: Apr. 28, 2026. [Online]. Available: https://www.postgresql.org/docs/current/tutorial-transactions.html

[16] "Indexes," postgresql. Accessed: Apr. 28, 2026. //url-https://www.postgresql.org/docs/current/indexes.html.

[17] "Sqlalchemy documentation," SQLAlchemy. Accessed: Apr. 28, 2026. [Online]. Available: https://docs.sqlalchemy.org/en/latest/

[18] M. Bayer, "Alembic documentation," Alembic. Accessed: Apr. 28, 2026. [Online]. Available: https://alembic.sqlalchemy.org/en/latest/

[19] "Introduction to windows service applications," Microsoft. Accessed: Apr. 19, 2026. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications

[20] "Guid function," Microsoft. Accessed: Apr. 2, 2026. [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-guid

[21] "Access control overview," Microsoft. Accessed: Apr. 19, 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows/security/identity-protection/access-control/access-control

[22] W. Eddy, Ed. MTI Systems, "Transmission control protocol (tcp)," Internet Engineering Task Force (IETF), Tech. Rep., Aug. 2022. [Online]. Available: www.rfc-editor.org/rfc/rfc9293.html

[23] "Powercfg command-line options," Microsoft. Accessed: Apr. 19, 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/design/

device-experiences/powercfg-command-line-options#option_waketimers

[24] "WMI start page," Microsoft. Accessed: Apr. 22, 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page

[25] C. Mutimukwe, O. Viberg, C. McGrath, and T. Cerratto-Pargman, "Privacy in online proctoring systems in higher education: stakeholders' perceptions, awareness and responsibility," *Journal of Computing in Higher Education*, 2025. [Online]. Available: https://link.springer.com/article/10.1007/s12528-025-09461-5

[26] K. Ball, "Electronic monitoring and surveillance in the workplace: Literature review and policy recommendations," Publications Office of the European Union, Luxembourg, Tech. Rep. JRC125716, 2021. [Online]. Available: https://publications.jrc.ec.europa.eu/repository/handle/JRC125716

[27] European Parliament and Council of the European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (general data protection regulation)," May. 4, 2016, chapter 2, Article 5(1)(c). [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[28] M. L. Cummings, "Automation bias in intelligent time critical decision support systems," in *AIAA 1st Intelligent Systems Technical Conference*. American Institute of Aeronautics and Astronautics, Sep. 2004. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2004-6313

A

Appendix: Alert Processing Flow

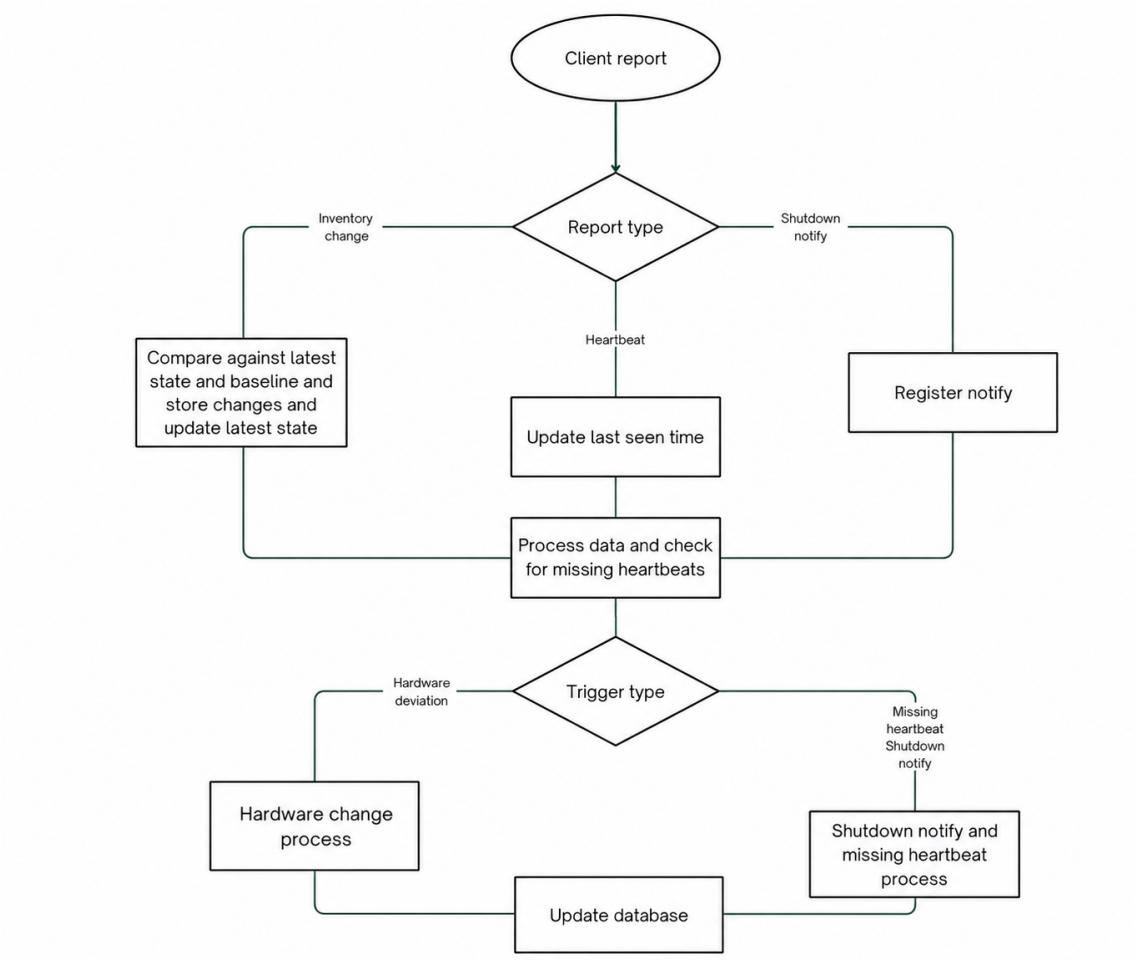

Figure A.1: Detailed alert-processing flow used by the prototype, see figure A.2 for hardware deviation process and figure A.3 for shutdown notify and missing heartbeat process

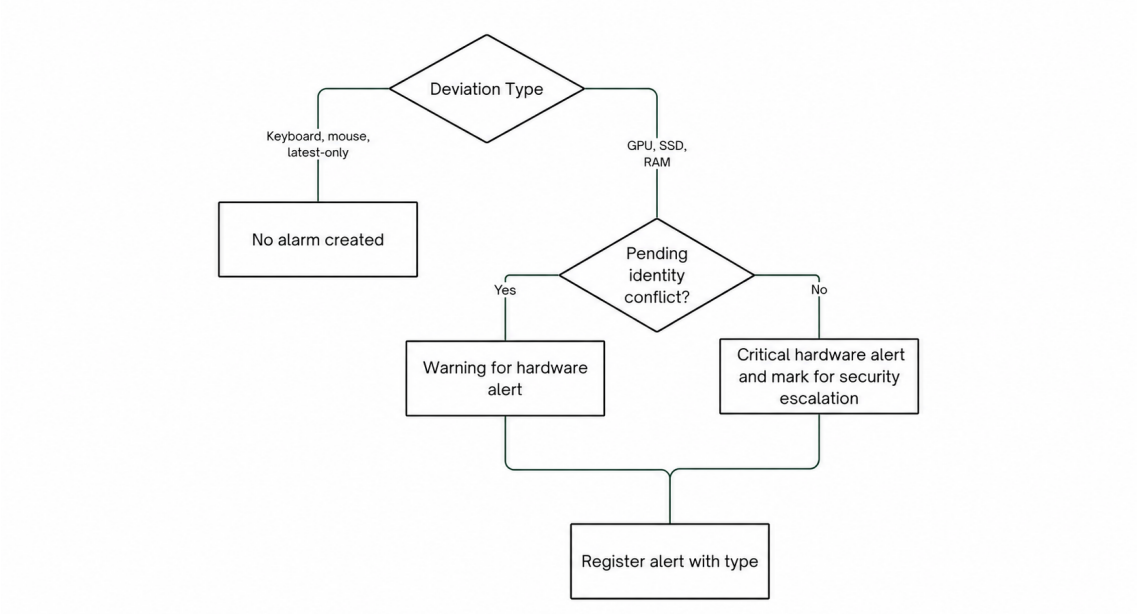

Figure A.2: The figure shows the "hardware change" process in detail

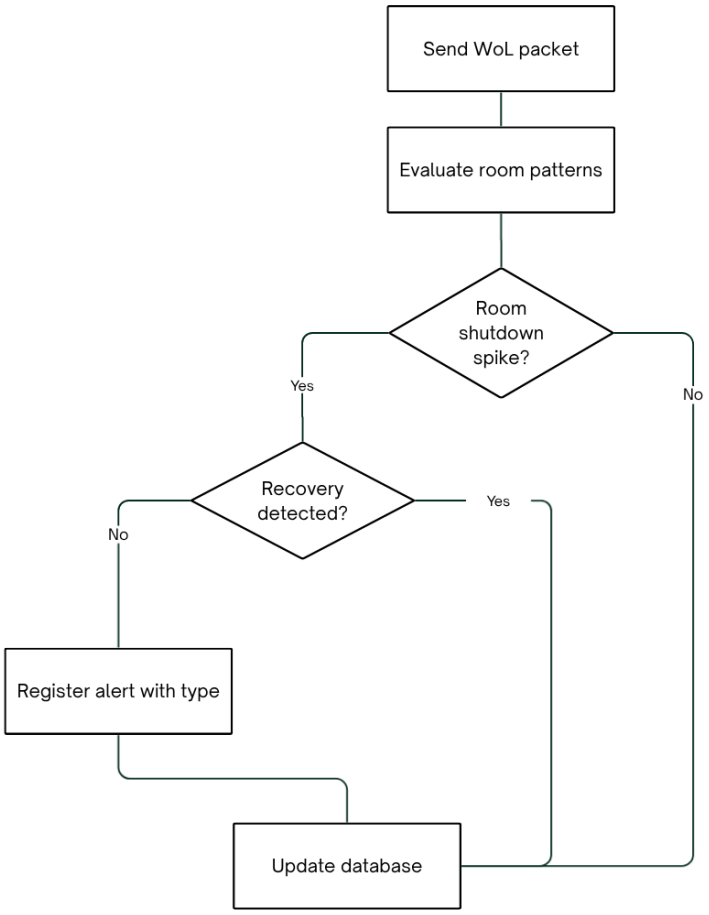

Figure A.3: The figure shows the "Shutdown notify and missing heartbeat" process in detail

B

Appendix: Database Schema

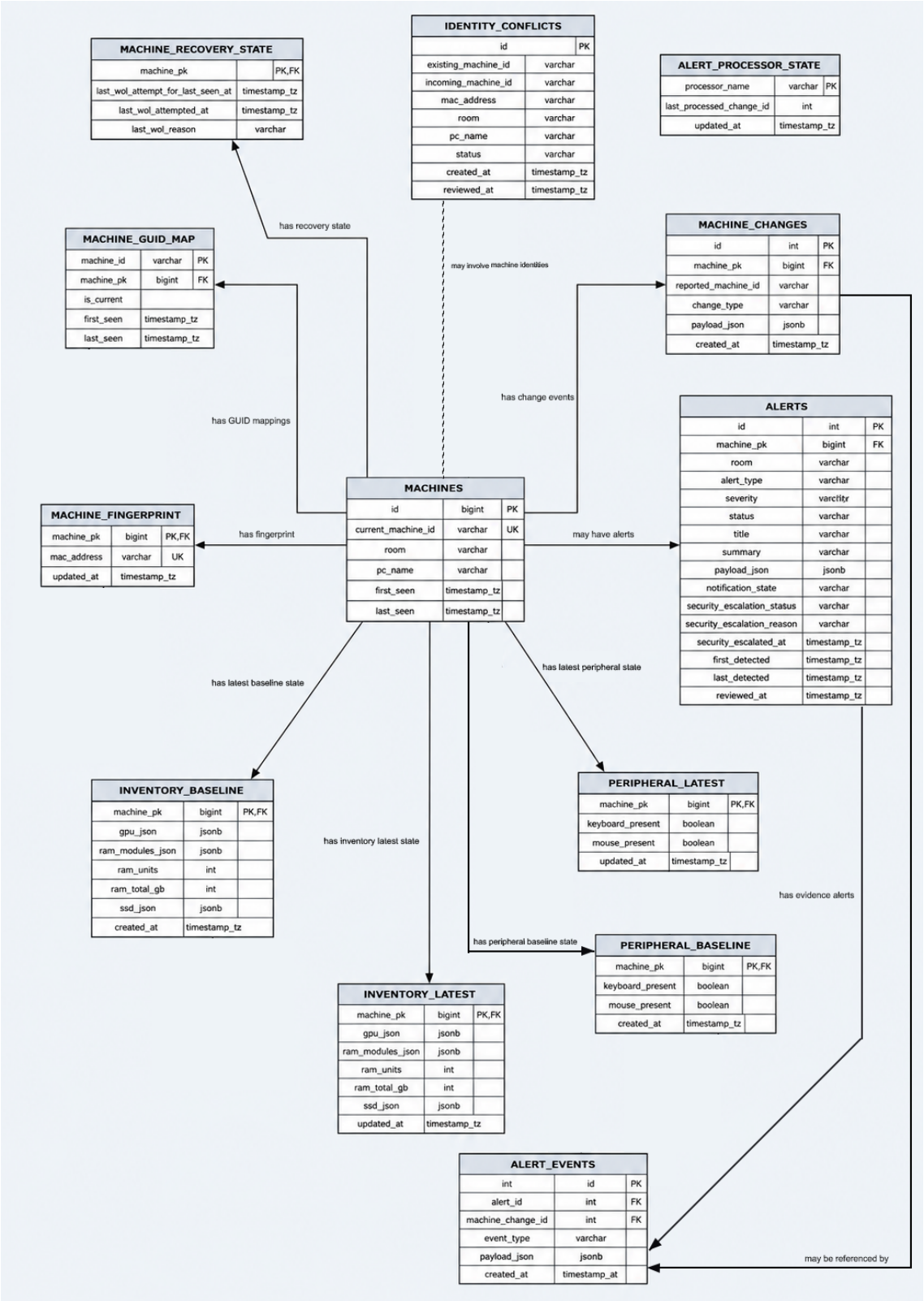

Figure B.1: Database schema showing the main tables used to store machine identity, inventory state, change events, alerts, and alert evidence.