: N° d'ordre
: N° de série

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research

UNIVERSITY OF ECHAHID HAMMA LAKHDAR - EL OUED

FACULTY OF EXACT SCIENCES
Computer Science Department

Thesis
submitted in partial fulfilment of the requirements for the degree of

ACADEMIC MASTER

Field: **Mathematics and Computer Science**
Option: **Computer Science**
Specialty: **Internet of Things & Cybersecurity**

Title

Design and Development of a Smart Access Control System based on Clark Wilson Security Model

Defended on June 22 th 2025

Submitted by :

- **Ahmed Siradj Eddine BEKKARI**
- **Hadjer AZZI**
- **Oussama KHECHEKHOUCHE**
- **Saif eddine MOUHADJER**

Examination Commitee :

Dr.	……	Professor	Chair
Dr.	……	MCA	Examinator
Dr.	**Mouhammed Mounir BOUHAMED**	MCB	Supervisor
Dr.	**Abdennacer KHELAIFA**	MCB	Supervisor

Academic year: 2024-2025

Abstract

Traditional Physical access control systems for sensitive facilities rely on outdated technologies vulnerable to identity spoofing, credential misuse, and tailgating attacks. These systems typically lack formal security models, creating significant gaps in operational integrity and policy enforcement that can lead to serious security breaches. This thesis aims to design and develop an intelligent physical access control system that adapts the Clark-Wilson integrity model for physical security environments, integrating formal policy enforcement with advanced biometric authentication and real-time monitoring capabilities.

Firstly, we extend and reformulate the Clark-Wilson model for physical access control through a novel decomposition into distinct Execution and Decision Environments. Moreover, we implement this adaptation as a dedicated software library using Test-Driven Development (TDD) methodology, providing an agile alternative to traditional formal methods. Secondly, over Clark-Wilson security model, we build our physical access control system that integrates two phases of dual biometric verification (facial recognition via DeepFace and fingerprint scanning) with YOLO-based people counting for comprehensive identity verification and tailgating prevention. We follow maintainable Aspect-Oriented Programming (AOP) to develop system components.

The implemented system successfully combines formal security model enforcement with practical biometric authentication and surveillance capabilities. A dual-phase user confirmation mechanism ensures integrity throughout the access lifecycle, while real-time presence monitoring enables anomaly detection and comprehensive access auditing. The system addresses critical vulnerabilities in traditional access control by providing verifiable policy enforcement beyond conventional system capabilities.

Key Words:
Physical Access Control Systems, Test-Driven Development, Clark-Wilson security model, Aspect-Oriented Programming, Integrity.

Résumé

Les systèmes traditionnels de contrôle d'accès physique pour les installations sensibles reposent sur des technologies traditionelle vulnérables aux attaques par usurpation d'identité, mauvaise utilisation des identifiants et attaques par suivi. Ces systèmes manquent généralement de modèles de sécurité formels, créant des lacunes importantes dans l'intégrité opérationnelle et l'application des politiques, ce qui peut entraîner des violations de sécurité graves. Ce Mémoire vise à concevoir et développer un système de contrôle d'accès physique intelligent qui adapte le modèle d'intégrité Clark-Wilson aux environnements de sécurité physique, en intégrant l'application formelle des politiques avec une authentification biométrique avancée et des capacités de surveillance en temps réel.

Premièrement, nous étendons et reformulons le modèle Clark-Wilson pour le contrôle d'accès physique à travers une décomposition novatrice en environnements distincts d'exécution et de décision. De plus, nous implémentons cette adaptation sous forme de bibliothèque logicielle dédiée en utilisant la méthode de développement pilotée par les tests (TDD), offrant une alternative agile aux méthodes formelles traditionnelles. Deuxièmement, sur le modèle de sécurité Clark-Wilson, nous construisons notre système de contrôle d'accès qui intègre deux phases de vérification biométrique (reconnaissance faciale via DeepFace et scan des empreintes digitales) avec un comptage de personnes basé sur YOLO pour une vérification complète de l'identité et la prévention des attaques par suivi. Nous utilisons la programmation orientée aspect maintenable pour développer les composants du système.

Le système implémenté combine avec succès l'application d'un modèle de sécurité formel avec une authentification biométrique pratique et des capacités de surveillance. Un mécanisme de confirmation d'utilisateur en deux phases garantit l'intégrité tout au long du cycle d'accès, tandis que la surveillance de présence en temps réel permet la détection d'anomalies et un audit complet des accès. Le système répond aux vulnérabilités critiques des systèmes traditionnels de contrôle d'accès en fournissant une application vérifiable des politiques au-delà des capacités des systèmes conventionnels.

Mots-clés :
Systèmes de Contrôle d'Accès Physiques, Développement Dirigées par les Tests, Modèle de Sécurité Clark-Wilson, Programmation Orientée Aspect, Intégrité.

الملخص

تعتمد أنظمة التحكم في الوصول التقليدية في المرافق الحساسة على تقنيات قديمة، ما يجعلها عرضة لانتحال الهوية، وسوء استخدام البيانات، وهجمات التسلل. وغالبًا ما تفتقر هذه الأنظمة إلى نماذج أمنية رسمية، مما يخلق فجوات كبيرة في السلامة التشغيلية وتطبيق السياسات، وقد يؤدي ذلك إلى انتهاكات أمنية خطيرة .تهدف هذه المذكرة إلى تصميم وتطوير نظام ذكي للتحكم في الوصول، يتوافق مع نموذج النزاهة "كلارك-ويلسون" في بيئات الأمان المادي، ويجمع بين تطبيق السياسات الرسمية، والمصادقة البيومترية المتقدمة، إضافة إلى قدرات المراقبة في الوقت الحقيقي .نقترح توسيعًا وإعادة صياغة لنموذج كلارك-ويلسون للتحكم في الوصول المادي، من خلال تقسيم مبتكر إلى بيئتين متميزتين: بيئة تنفيذ، وبيئة اتخاذ القرار. أولًا، يتم تنفيذ هذا التكيف في شكل مكتبة برمجية مخصصة باستخدام منهجية TDD، لتوفير بديل مرن للطرق الرسمية التقليدية .ثانيًا، بالاستناد إلى نموذج كلارك-ويلسون الأمني، نقوم ببناء نظام للتحكم في الوصول يدمج مرحلتين من التحقق البيومتري المزدوج: التعرف على الوجه باستخدام تقنيةDeepFace ، ومسح بصمات الأصابع. ويُضاف إلى ذلك استخدام تقنية YOLO لعدّ الأشخاص، من أجل التحقق الشامل من الهوية ومنع التسلل. ويُطوّر النظام باستخدام أسلوب AOP . يجمع النظام المطوّر بين تطبيق رسمي للنموذج الأمني، وتقنيات المصادقة البيومترية العملية، وقدرات المراقبة الفورية. وتُوفر آلية التحقق الثنائي ضمانًا للسلامة طوال دورة الوصول، بينما تتيح المراقبة في الوقت الحقيقي الكشف عن السلوكيات الشاذة وتدقيق عمليات الوصول بدقة. ويُعالج هذا النظام الثغرات الجوهرية في أنظمة التحكم التقليدية، من خلال تقديم تطبيق يمكن التحقق منه للسياسات الأمنية، يتجاوز قدرات الأنظمة التقليدية

الكلمات المفتاحية:

أنظمة التحكم في الوصول المادي، TDD، نموذج الأمان كلارك-ويلسون، AOP، السلامة.

Acknowledgements

We would like to extend our heartfelt appreciation to all those who contributed to the success of this project.

First and foremost, our sincere gratitude goes to our supervisors, DR. Mohammed Mounir Bouhamed and DR. Abdennacer Khelaifa and Dr. Ahmed Ghanabzia , for their exceptional guidance, constant encouragement, and valuable insights throughout every stage of this work. Their mentorship has been crucial not only in refining our ideas but also in helping us grow as learners and professionals.

We thank the LIAP Laboratory and the House of Artificial Intelligence, especially their director, Professor Laouid Abdelkader, for his assistance and for providing us with some tools and resources to complete the project. We also thank the Business Incubator of the University of El Oued for their training and support in obtaining the 'Startup Label' and submitting a patent application. We would like to thank Professor Samir Kaddouri, as well as Dr. Djirbi Rachid from the University of Boumerdes.

To our families and friends, thank you for your unwavering support, patience, and motivation during the challenges and milestones of this journey.

Finally, we extend our appreciation to each other as teammates—Ahmed Siradj Eddine Bekkari, Hadjer Azzi, Oussama Khechekhouche, and Saif eddine Mouhadjer—for the spirit of collaboration, shared responsibility, and mutual respect that made this project not only possible but enjoyable.

This project has been a meaningful learning experience for all of us, and we are truly grateful to everyone who played a part in it.

Contents

List of Figures

List of Tables

List of Listings

General Introduction

Protecting sensitive facilities is crucial amid rapid technological progress and security threats. The IoT enhances physical security through real-time monitoring, automation, and intelligent decisions. Intelligent access control systems are vital for safeguarding personnel and property. However, traditional systems, often using outdated technologies, are vulnerable to identity spoofing, credential misuse, and delayed responses to unauthorized activities.

Various smart security systems leverage AI and IoT to enhance surveillance and access control. Examples include IoT-based systems in Smart University laboratories [2], biometric fingerprint systems for presence checking and room access [47], smart attendance and leave management systems using fingerprint recognition in academic institutions [37], and RFID and fingerprint technologies for securing smart buildings [5]. The mining sector requires rigorous access management to control activities and restrict unauthorized entry [31].

Despite these advancements, many systems have critical flaws, especially those relying on RFID cards, keypads, or magnetic tokens. These credentials verify possession, not holder identity [23], allowing identity spoofing, credential sharing, and unauthorized access.

Moreover, these systems often lack a formal access control security model, hindering operational integrity and permission misuse prevention [27]. Many systems lack robust mechanisms to verify if an individual is authorized or has entered alone, allowing tailgating or piggybacking [54].

These gaps make systems vulnerable to manipulation and serious security breaches. An incident at Oran Airport in 2024 demonstrated this vulnerability: a young man reportedly infiltrated an Air Algérie aircraft [7]. This event highlighted the need for more robust, intelligent surveillance and access control.

To bridge these gaps, this research proposes an intelligent, real-time system addressing key vulnerabilities. It integrates dual biometric verification (fingerprint and facial recognition) using computer vision and IoT. Unlike traditional approaches, it adapts the Clark-Wilson security model for physical access control, ensuring strict policy enforcement and data integrity. This multi-layered authentication is a distinguishing feature. The goal is to enhance physical security through automated identity verification, intelligent decisions, and secure, policy-driven access control.

This research contributes to intelligent surveillance and access control by: adapting the Clark-Wilson security model for physical security environments, facilitating formal policy enforcement beyond traditional system capabilities; using TDD for the model's implementation, enhancing practical applicability; featuring robust dual biometric verification (fingerprint, facial recognition) at entry and within the facility; and incorporating a real-time, YOLO-based people counter. These components form an integrated, intelligent framework addressing critical gaps in current systems.

Dual biometric verification is central to the intelligent surveillance system, implemented at the entrance gate and inside the facility. The system combines fingerprint

recognition with facial recognition using the DeepFace model. Real-time internal facility monitoring using YOLO further enhances system capabilities. This detects and counts individuals, verifying if an authorized person is accompanied by an unauthorized individual. This multi-layered approach ensures identity verification integrity and prevents access privilege misuse.

This thesis contributes to secure system design by:

1. Extending the Clark-Wilson security model to physical systems, enabling secure policy enforcement in real-world environments.
2. Reformulating the Clark-Wilson architecture to facilitate integration with hybrid digital-physical security frameworks.
3. Decomposing the Clark-Wilson model into distinct Execution and Decision Environments for clearer separation of operational logic and policy enforcement.
4. Adopting TDD as an agile alternative to traditional formal methods (FM) for modeling and testing Clark-Wilson-based systems.
5. Applying an approach based on **Aspect-Oriented Programming** [26] for developing a maintainable architecture and separate decision library from execution environment.
6. Introducing a dual-phase user confirmation mechanism (pre-entry and post-entry) to ensure **integrity** throughout the access lifecycle.
7. Integrating dual biometric authentication (facial recognition, fingerprint scanning) for enhanced, robust identity verification.
8. Using people-counting technologies for real-time occupancy monitoring, anomaly detection (e.g., tailgating), and comprehensive access auditing.

In addition to the contributions outlined above, this research has led to a formal request for a patent and the creation of a startup label aimed at commercializing the proposed smart access control system, further advancing its application in real-world environments.

This thesis is organized as follows:

- **Chapter 1** introduces key concepts in access control, biometric authentication, IoT-based surveillance, and the Clark-Wilson integrity model.
- **Chapter 2** describes developing a software library based on an adapted Clark-Wilson model using TDD to ensure secure, verifiable, and robust control.
- **Chapter 3** details integrating the adapted Clark-Wilson model into a smart access control system, covering biometric verification, YOLO-based people counting, and IoT component communication protocols.

Chapter 1

Background and Related works

1.1 Introduction

Technological advancements progress rapidly. Thus, security systems, particularly access control solutions, are commonly deployed in diverse environments, including residences, workplaces, and financial institutions, enhancing societal safety and asset protection. The IoT and AI advancements have accelerated this evolution, enabling smarter, more context-aware access control decisions.
Despite these innovations, traditional access mechanisms are often inadequate, failing to ensure data integrity and prevent unauthorized access. Thus, a critical question is: How can modern access control systems' integrity and security be established and maintained ?
Security models offer frameworks to address this. The Clark-Wilson integrity model, e.g., is vital, maintaining data accuracy and enforcing strict control through well-defined access policies. These policies permit only authorized users to perform specific, validated operations on designated data items.
This chapter provides background on:

- **Smart Access Control Systems:** Their evolution, components, and AI/IoT's role.
- **The Clark-Wilson Security Model:** Its principles, components, and relevance to data integrity in access control.
- **Test-Driven Development:** Its methodology and benefits for developing reliable, verifiable software.
- **Related Work:** Existing research and solutions in intelligent access control and security model application.

1.2 Surveillance systems

This section reviews surveillance system evolution, general architecture, the transition to smart surveillance with IoT and AI, and applications of these advanced systems.

1.2.1 History and evolution

Surveillance systems evolved significantly with technology and increasing societal security needs. Early analog systems, CCTV, captured footage on videotapes, requiring manual monitoring. These systems had limited storage, scalability, and real-time response capabilities [49]. The early 2000s saw a significant evolution with digital video recording and IP cameras. This pivotal development enabled clearer image capture,

remote footage access, and more efficient storage [67], enhancing system flexibility, accessibility, and integration of advanced analytics [14]. AI is transforming surveillance. Computer vision and deep learning empower systems to detect objects, recognize faces, analyze behavior, and identify anomalies in real-time [34]. This shift transforms surveillance from a passive tool to an active, data-driven security approach, aligning with this paper's research objectives.

1.2.1.1 General architecture of surveillance systems

A typical surveillance system comprises interconnected components that capture, transmit, store, and display video footage. Design varies with system complexity and application, but core elements include cameras, storage devices (e.g., DVRs, NVRs), monitors, and networking infrastructure [69]. Key components include:

- **Cameras:** Primary visual input devices capturing video footage; can be analog (CCTV) or digital (IP-based) depending on system configuration and requirements.
- **Transmission Network:** Transmits video data from cameras to storage or processing units using technologies like coaxial cables (older analog systems) or Ethernet, Wi-Fi, and fiber optics (modern digital systems).
- **Recording and Storage Devices:** DVRs (analog systems) or NVRs (IP-based systems) store video footage for review, analysis, or evidence.
- **Monitoring Unit:** A user interface (dedicated monitor or control software) allowing security personnel to view live footage, review recorded events, and analyze events in real-time.

These components form the foundational architecture of most contemporary surveillance systems, which can be augmented with advanced analytics like facial recognition and object detection via software upgrades or additional processing units [62]. Figure 1.1 present an example of Surveillance System [19]

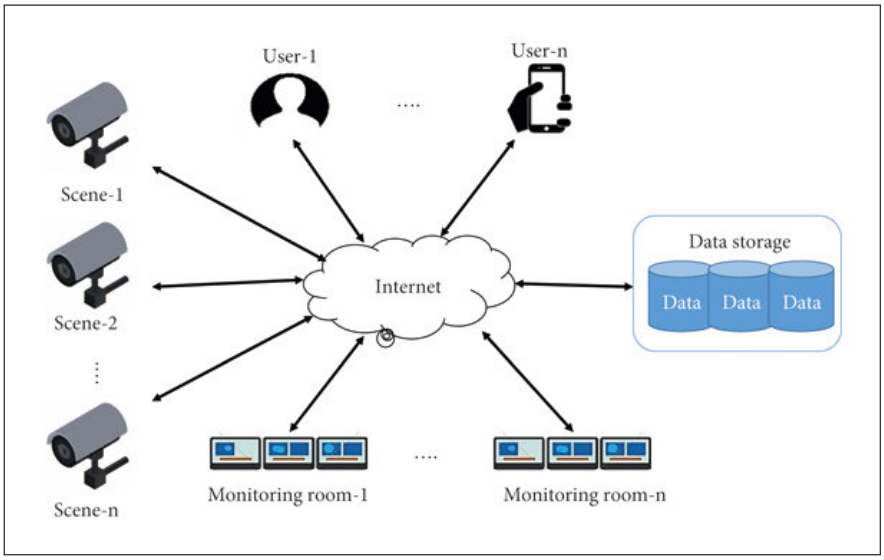


FIGURE 1.1: A typical architecture of surveillance system

1.2.2 Smart surveillance transition

Traditional surveillance systems, mainly CCTV, operated passively, heavily relying on human operators to monitor video feeds and identify security incidents or operational issues. This reliance often caused delayed reactions and operator fatigue during extended monitoring [48]. Recent advancements in computing, networking, and artificial intelligence have significantly enhanced system capabilities. Modern surveillance systems are more intelligent, performing real-time video analysis and sometimes autonomous decision-making. Advanced systems use object detection, facial recognition, motion tracking, and behavioral analysis to proactively identify threats and anomalies, enabling preventative measures or rapid responses to mitigate escalation [70]. Smart surveillance systems offer several advantages over traditional ones [44]:

- **Automation:** AI-driven alerts and event classifications reduce constant human oversight, allowing personnel to focus on verified incidents.
- **Accuracy:** Deep learning algorithms, when properly trained and deployed, generally improve detection accuracy for specific tasks, reducing errors like false alarms.
- **Scalability:** Cloud and edge computing architectures facilitate system expansion, remote management, and distributed access to surveillance data and functions.
- **Real-Time Response:** Automated detection and alerting enable immediate action to unusual or critical events, minimizing manual monitoring delays.

This transition from a reactive to a proactive security paradigm significantly changes the design philosophy and operational application of surveillance systems [35].

1.2.2.1 Internet of Things in Surveillance Systems

The Internet of Things is a framework where physical objects (things)—devices, machines, sensors, actuators—are embedded with sensors, software, and communication technologies [1, 4]. These interconnected things autonomously collect data, interact with their internal states and external environments, and transmit information across a network, often without direct human intervention. IoT enables diverse real-time, data-driven applications and services. IoT systems typically use a layered architecture: perception, network, and application layers [4]. Core IoT functionality often involves M2M communication and seamless cloud platform integration, bridging the physical and digital worlds [29]. This broader IoT integration allows creating smart environments in healthcare, manufacturing, agriculture, transportation, and smart cities.

- **The Three-Layer IoT Architecture:** This architecture, introduced by Ait et al. [4] and widely adopted. The Figure 1.2 depicted this architecture, typically includes:
  - *Perception Layer:* This physical layer incorporates sensors and actuators to gather environmental information (e.g., temperature, movement, images) or effect changes.
  - *Network Layer:* Connects devices and transmits collected data to other devices, gateways, or servers for processing, using various communication protocols.

– *Application Layer:* Processed data provides value-added services to end-users or systems, encompassing smart homes, smart cities, and smart healthcare solutions.

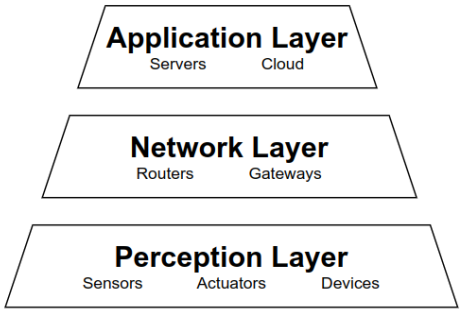


FIGURE 1.2: The three layers architecture.

- **IoT System Components:** An IoT system integrates technological components for smart connectivity and automation (Figure 1.3, based on Afouf et al. [2]). Key components include connected devices (sensors, actuators) interacting with the physical environment. These devices transmit data via wireless networks (e.g., Wi-Fi, Bluetooth, LoRaWAN, NB-IoT) to platforms for aggregation, storage, and processing, often using cloud infrastructure. Such systems enable real-time analysis, remote monitoring, and automated control across domains.

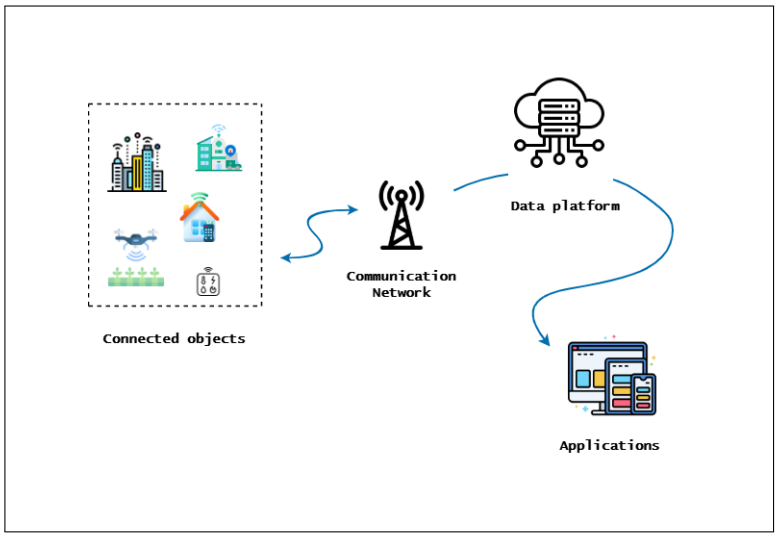


FIGURE 1.3: IoT system components

- **Connected Objects (IoT Devices):** IoT devices are physical items with embedded sensors, software, and communication technologies enabling data collection, processing, and exchange with other devices and systems [2]. These devices sense their environment, act on instructions or pre-programmed logic, and communicate via protocols like Wi-Fi, Bluetooth, Zigbee, or MQTT. They are widely used in security, healthcare, industry, and smart homes.

- **MQTT Communication Protocol:** MQTT is a prominent, lightweight, and efficient communication protocol often used in IoT system development [33].

  1. *MQTT Protocol Overview:* MQTT is a lightweight, open-standard, efficient messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. This makes it suitable for M2M communication and diverse IoT applications [33].
  2. *MQTT Protocol Architecture:* MQTT uses a client-server architecture with a publish/subscribe messaging pattern. Clients (devices or applications) exchange messages via a central message broker (Figure 1.4). Messages are published to "topics" (named message channels). Clients subscribe to topics to receive messages published to those channels. [33] The pub-

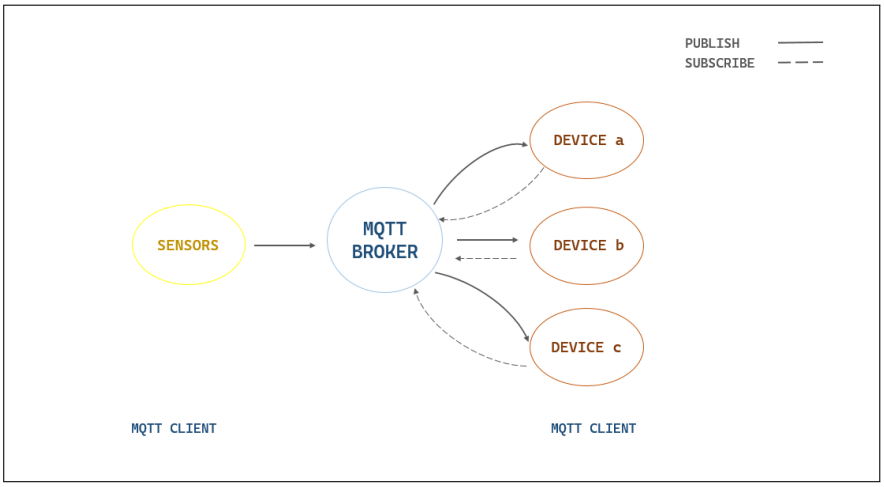


FIGURE 1.4: MQTT Protocol Architecture

     lish/subscribe model decouples message producers from consumers:
     - *Publish:* A client (publisher) sends a message to a topic on the broker, without knowing which clients will receive it.
     - *Subscribe:* Clients (subscribers) inform the broker of interest in a topic (or topics using wildcards) and receive all messages published to it.
  3. *MQTT Broker:* The central server component managing message routing between clients. It receives messages from publishers and forwards them to subscribers of relevant topics, ensuring delivery per specified QoS levels. Widely used brokers include Mosquitto [46] and HiveMQ [32].
  4. *MQTT Client:* Any device or application program connecting to an MQTT broker [33]. A client can be a publisher, subscriber, or both. Examples range from sensors and microcontrollers to mobile applications and enterprise software.

- **IoT Challenges:** Despite benefits, widespread IoT system implementation presents significant challenges requiring careful consideration and mitigation [58]:

  - Security and privacy risks from increased attack surfaces and sensitive data collection by many connected devices.
  - Scalability concerns in managing and supporting exponentially increasing interconnected devices and their data.

- Ensuring interoperability among diverse devices, communication protocols, and platforms from different manufacturers.
- Managing, processing, and deriving insights from large volumes of IoT-generated data in real-time or near real-time.

1.2.2.2 Artificial Intelligence in Surveillance

AI is pivotal for enhancing modern surveillance systems, transforming them from reactive observation tools into intelligent, proactive security platforms. Historically, surveillance relied on human operators manually monitoring video feeds—a time-consuming, error-prone process subject to attention fatigue. AI-powered systems automate many surveillance tasks using advanced machine learning and deep learning algorithms to process and analyze vast real-time video data streams. Such systems autonomously detect objects, recognize faces, track behaviors, or identify anomalous events, often without initial direct human intervention [55, 71].

- **Key AI Techniques in Surveillance:** AI surveillance techniques, broadly classified into key categories, significantly enhance monitoring and analytics(1.5):
  - **Object Detection:** Models like YOLO [52] and SSD (Single Shot MultiBox Detector) [39] identify and locate objects (e.g., humans, vehicles, bags) in live or recorded video.
  - **Facial Recognition:** Systems like FaceNet [55] and DeepFace [60] generate unique numerical facial representations (embeddings), enabling real-time individual verification (one-to-one) and identification (one-to-many).
  - **Activity and Anomaly Detection:** AI models trained to recognize normal behavior patterns and detect deviations indicating unusual activities (e.g., fighting, loitering, crowd panic) [59].
  - **Crowd Analysis:** Techniques estimate crowd density, track movement, and identify dangerous crowd behaviors, aiding event management and preventing hazardous overcrowding [72].
  - **License Plate Recognition (LPR):** Often combines OCR with CNNs for automated vehicle license plate identification, supporting smart traffic enforcement and vehicle tracking.

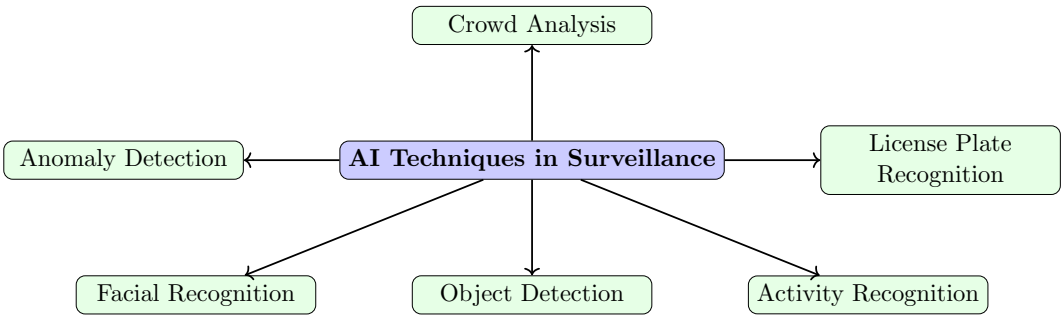


FIGURE 1.5: Main AI Technique Categories in Surveillance.

AI integration has made modern surveillance significantly more effective, scalable, and intelligent. AI-driven systems using object detection, facial recognition, and anomaly detection monitor environments with greater real-time autonomy and precision, reducing human operator cognitive load. These improvements enhance community safety and security and provide sophisticated methods to identify and address potential threats before escalation. However, deploying AI in surveillance presents persistent challenges, particularly ethical concerns of privacy, algorithmic bias, decision-making transparency, and system adaptability to diverse, dynamic real-world environments [11].

1.2.3 Smart surveillance system applications

Widespread deployment of intelligent, networked devices has increasingly interconnected people, data, and processes. Notable applications include maritime operations [6], healthcare [20, 38], agriculture [22, 56], and smart city development [57]. In urban environments, IoT facilitates applications like intelligent parking management, adaptive smart lighting, and real-time public surveillance. Human oversight collaborating with automated device capabilities enhances safety and operational efficiency, particularly during emergencies [15]. Modern surveillance systems, using smart cameras and AI-powered facial recognition, significantly aid threat detection and identifying individuals of interest. In the personal sphere, IoT enables more accessible health self-management via wearable devices and ambient environmental sensors. These advancements support safer, healthier living and working spaces.

1.2.3.1 Smart cities and public safety

IoT initiatives are crucial for developing and enhancing smart city infrastructures. Figure 1.6 shows IoT's technological development and future predictions in smart city paradigms (Sharma et al. [57]).

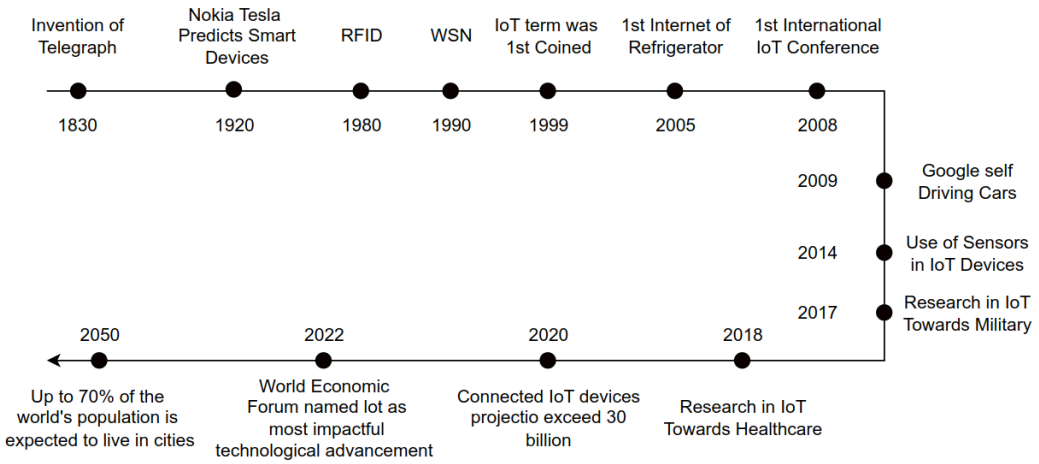


FIGURE 1.6: IoT evolution in smart cities, adapted from

Key research and applications include: Chatzimichail et al. [15] explored how IoT technology can be used to improve public safety by creating a connected infrastructure in public areas. Their system brings together smart devices and cloud-based analytics to detect threats in real time and help coordinate emergency responses. This makes it particularly useful in places like city squares or parks where quick action is essential.

In a similar direction, Lulla et al. [40] developed a smart surveillance and security system using IoT sensors and cameras. Their design focuses on keeping urban areas under watch in a more organized and responsive way. Like the previous study, this system emphasizes real-time monitoring, but with a tighter focus on combining multiple hardware elements for more thorough coverage.

Cob et al. [18] took a different approach by proposing a video surveillance system based on edge computing. Instead of relying heavily on cloud services, their system processes data locally, which helps reduce delays and limits bandwidth usage. While it also targets smart city applications like Lulla et al., it offers a faster and more efficient way to handle large amounts of video data.

Sharma et al. [57] provided a broader view through a survey of IoT-based smart surveillance systems. Their work highlights recent progress in the field, outlines different use cases, and shows how technology is evolving to better support public safety. Though it doesn't present a specific system, it helps connect the dots between different approaches, including those mentioned above.

Patil et al. [50] introduced a more accessible solution using Raspberry Pi to create a low-cost surveillance system. It supports real-time motion detection and image capture, making it a good fit for homes or small-scale environments. While it shares the goal of real-time monitoring with larger systems, it stands out for its simplicity and affordability.

Paul et al. [51] focused on automation by designing an IoT-based surveillance framework that responds to threats with minimal human input. Their system detects issues and acts immediately, helping reduce delays and manual oversight. Like Chatzimichail et al., their work targets fast response, but with a stronger push toward fully automated decision-making.

Finally, a general overview in [25] looks at smart home surveillance, including intrusion detection and remote access. This aligns with Patil et al.'s focus on smaller environments but broadens the scope to cover more advanced features for home security.

1.2.3.2 Healthcare Surveillance

In the healthcare sector, surveillance systems are increasingly supported by machine learning and artificial intelligence to enhance patient care and safety. These systems are commonly used in hospitals and elderly care centers to monitor patient behavior, detect anomalies, and even interpret emotional states. Dhuheir et al. [20] explored emotion recognition technologies in healthcare surveillance, highlighting how AI can help caregivers better understand patient conditions and respond appropriately. In addition to emotional monitoring, intelligent health systems are being developed for more targeted applications. For instance, Lavanya et al. [38] proposed a predictive cardiovascular monitoring system based on the Internet of Medical Things (IoMT) and LoRa communication. Their system enables real-time tracking of heart-related health parameters, allowing for early detection of risks and timely medical intervention.

1.2.3.3 Agricultural Surveillance

In agriculture, the growing global demand for food has driven the adoption of smart surveillance technologies to improve productivity and ensure sustainability. One notable advancement is the use of drones equipped with AI to oversee large farming areas. Shah et al. [56] presented a drone-based agricultural surveillance system that uses convolutional neural networks (CNNs) and Android applications to monitor crop health, detect pest infestations, and track field activity. Beyond crops, surveillance also extends to protecting farms from animal-related threats. Doshi et al. [22] described a system for monitoring the movement of animals and birds in agricultural zones, aimed at minimizing crop damage and supporting better farm management. These innovations demonstrate how AI-powered surveillance is reshaping both healthcare and agriculture, enabling smarter, more responsive environments.

1.2.3.4 Access control and high-security zones

Surveillance technologies are critical for maintaining security in restricted or sensitive areas where unauthorized access or undetected activity can pose serious risks. In this context, Afreen et al. [3] proposed an integrated IoT-based surveillance system specifically designed to secure such zones. Their solution brings together multiple sensors and supports remote monitoring, allowing for continuous observation and timely response to potential threats. This approach enhances situational awareness and reinforces access control in high-risk or confidential environments.

1.2.3.5 AI-driven surveillance systems and techniques

AI-driven surveillance systems are reshaping the way public safety and monitoring are handled by introducing smarter, faster, and more scalable techniques. Zhao et al. [71] demonstrated the power of YOLOv3 for real-time object detection, showing how it can significantly improve the responsiveness of surveillance systems. By accurately identifying people and objects in real-time, this method supports faster decision-making and more effective threat analysis in public environments.

In a different but complementary direction, Schroff et al. [55] introduced FaceNet, a deep learning model that maps facial images into a compact Euclidean space. This allows for highly accurate face verification and clustering, which is especially valuable in large-scale surveillance systems where identifying and tracking individuals across multiple cameras or timeframes is essential.

Adding to the capabilities of intelligent monitoring, Sultani et al. [59] proposed a technique for detecting anomalies in surveillance videos using weak supervision. Their method leverages deep neural networks to learn from large datasets without requiring detailed annotations for every unusual event. This makes it more practical for real-world deployment, where labeling data can be time-consuming and inconsistent.

1.3 Access Control Systems

After explaining surveillance systems in the previous section, we now turn our attention to Access Control Systems (ACS). These systems are designed to restrict unauthorized entry to physical spaces or access to digital environments. Over time, ACS technologies have evolved from simple approaches—such as mechanical locks and basic passcodes—to more advanced electronic systems that incorporate sophisticated authentication methods, including biometric recognition [13].

1.3.1 Classic Access Control Systems

Classic Access Control Systems rely on traditional, simple authentication methods for securing physical and digital spaces. These systems typically use mechanical locks with physical keys, basic PIN codes, magnetic stripe cards, or simple password-based authentication. They operate on straightforward mechanisms with limited verification capabilities, often providing single-factor authentication and basic logging functionality. Classic systems are generally standalone solutions with minimal integration capabilities and require manual management for user access rights.

1.3.2 Modern Access Control Systems

Modern Access Control Systems employ advanced electronic technologies and sophisticated authentication methods to provide enhanced security and functionality. These systems integrate multiple authentication factors including biometric recognition (fingerprint, facial, iris scanning), smart cards with encryption, mobile credentials, and AI-powered analytics. They feature networked architectures enabling centralized management, real-time monitoring, automated access logging, and integration with other security systems. Modern systems offer scalable, flexible solutions with cloud connectivity, remote management capabilities, and adaptive security protocols that can respond dynamically to security threats.

1.3.3 Classic vs Modern Access Control Systems

Table 1.1 compares key features of classic (traditional) and modern ACS [12], reflecting common distinctions in security literature [12] regarding technology, operational characteristics, user experience, and security capabilities.

Feature	**Classic ACS**	**Modern ACS**
Authentication Methods	RFID cards, keypad passcodes, basic fingerprint readers.	Advanced biometrics (e.g., multi-modal, fingerprint/facial recognition with liveness detection), mobile credentials (e.g., smartphone apps using Bluetooth/NFC), QR codes.
Credential Storage	Local (e.g., standalone controllers, cards).	Cloud-based or hybrid (local device/controller storage with centralized cloud databases).
Accuracy/ Security	Moderate; susceptible to card duplication, lost/stolen cards, shared passcodes. Early biometrics could be spoofed.	High; advanced biometrics with liveness detection reduce spoofing. Encrypted mobile credentials enhance security over physical tokens.
User Experience	Requires manual interaction (e.g., swiping cards, keying passcodes), often slower.	Offers contactless, seamless access (e.g., walk-through facial recognition, tap-and-go mobile access), improving speed and convenience.

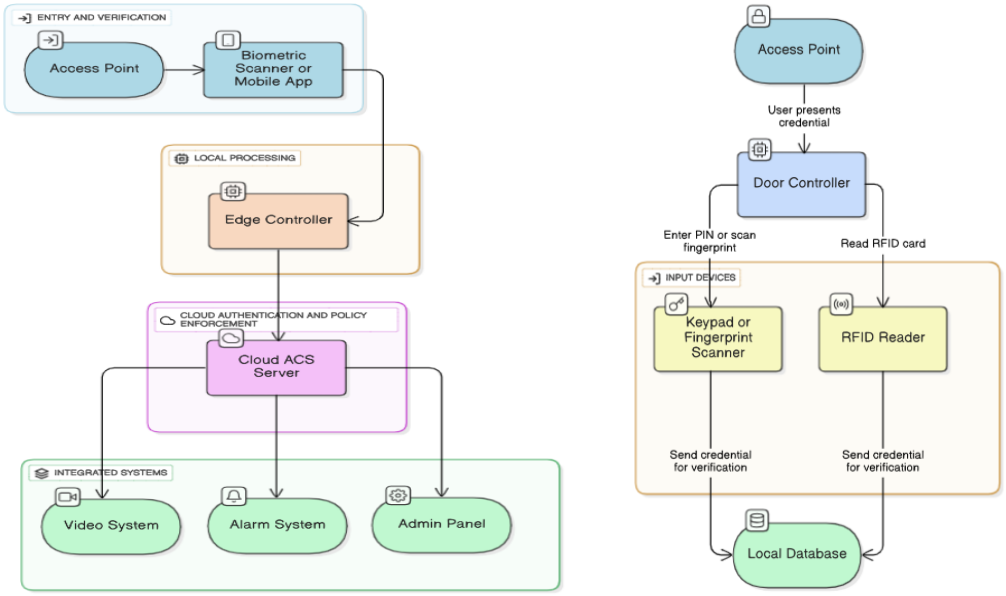
Continued on next page

Continued from previous page		
Feature	**Classic ACS**	**Modern ACS**
Maintenance	Frequent physical maintenance (e.g., card reader wear, keypad malfunctions, battery replacements).	Reduced physical reader wear (especially contactless); requires diligent software/firmware updates for security and features.
Integration Capabilities	Basic; limited to standalone door locks or rudimentary alarm panel connections.	Advanced, extensive integration with video surveillance, intrusion alarms, building management systems, identity management platforms, mobile applications, and IoT devices.

TABLE 1.1: Comparison between Traditional and ACS.

1.3.4 Architecture

Figures 1.7a and 1.7b illustrate conceptual configurations of modern and traditional ACS, respectively, showcasing their components and data flow. Modern ACS architectures increasingly integrate cloud services and IoT devices, while traditional ACS typically use localized, often standalone, hardware with limited external connectivity [30]. Figure 1.7b (Traditional ACS) shows predominantly hardware-centric architectures, often limiting integration and confining management to on-premises. Modern ACS architectures (Figure 1.7a) leverage cloud services for centralized management and data storage, mobile integration for credentialing and user interaction, and real-time data analytics for enhanced monitoring. These architectural advancements in modern ACS offer improved scalability for more users and access points, enhanced security through centralized updates and analytics, and greater user convenience via streamlined access [68].



(A) Modern ACS (B) Traditional ACS

FIGURE 1.7: Typical architectures of Modern and Traditional Access Control Systems.

1.4 Clark-Wilson Security Model

1.4.1 Definition

The **Clark-Wilson** model is a security model designed to preserve data **integrity** by implementing a system where all data changes occur through carefully controlled transactions. Limits users to performing only the transactions they are allowed to perform, ensuring that each transaction maintains the integrity of the system by moving data from one consistent state to another. [17].

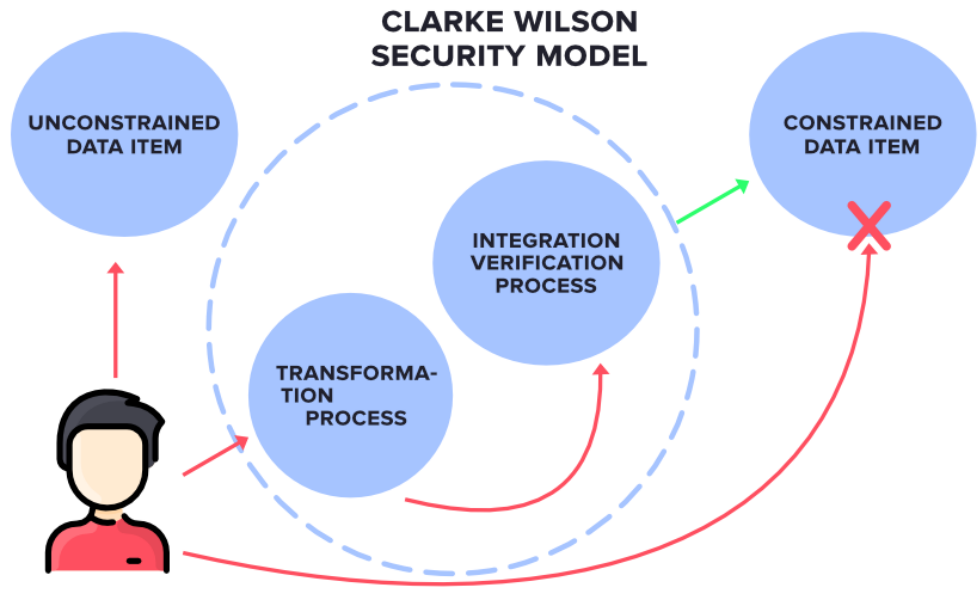


FIGURE 1.8: Conceptual Overview of the Clark-Wilson Security Model.

1.4.2 Key Principles & Core Concepts

The table below describes core concepts and key principles of the Clark-Wilson security modell.

Category	Concept	Description
Key Principles	Well-formed Transactions	Data are manipulated in restricted ways that preserve or ensure integrity, not arbitrarily. [16]
	Separation of Duties	Each user has access to only specific, limited sets of programs. [16]
Core Concepts	Transformation Procedures (TPs)	The only procedures allowed to modify CDIs; ensure transition from one valid state to another. [16]
	Integrity Verification Procedures (IVPs)	Procedures to verify the integrity of CDIs is maintained. [16]
	Constrained Data Items (CDIs)	Objects whose integrity is protected by the system. [16]
	Unconstrained Data Items (UDIs)	Objects not covered by the integrity policy. [16]

TABLE 1.2: Clark-Wilson Security Model: Key Principles and Core Concepts

1.4.3 Policy Rules

The Clark-Wilson model enforces data integrity through two sets of policy rules: Certification Rules (C-rules) and Enforcement Rules (E-rules).

1.4.3.1 Certification Rules

- **C1 (IVP Certification)** All IVPs must properly verify that CDIs are in a valid state whenever they are executed. [61]
  Example: Bank Account Balance Verification System
  Imagine a bank's ATM withdrawal system:
  - **CDI**: Customer account balance ($1,500)
  - **IVP**: Balance validation checker

  Scenario: Before any transaction occurs, the IVP runs automatically to verify:
  - Account balance is not negative
  - Account number exists in the system
  - Account is not frozen/suspended
  - Balance format is correct (proper currency format)

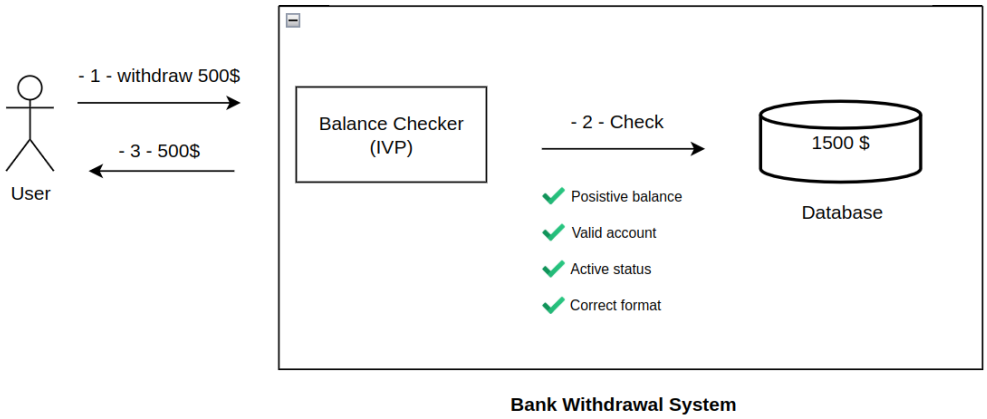


FIGURE 1.9: Bank Withdrawal System Example

- **C2 (Validity)** All TPs must be validated to ensure correctness. This means that if a CDI starts in a valid state, the TP must transform it into another valid state. Each TP must be certified for a designated set of CDIs by defining relations in the form $(TP_i, (CDI_a, CDI_b, ...))$. [61]
  Example: Online Banking Transfer System
  - **CDI1:** Sender's account balance: $1,000 (valid state)
  - **CDI2:** Receiver's account balance: $500 (valid state)
  - **TP (Transformation Procedure):** Money transfer function

  Scenario: Customer wants to transfer $200 from Account A to Account B.

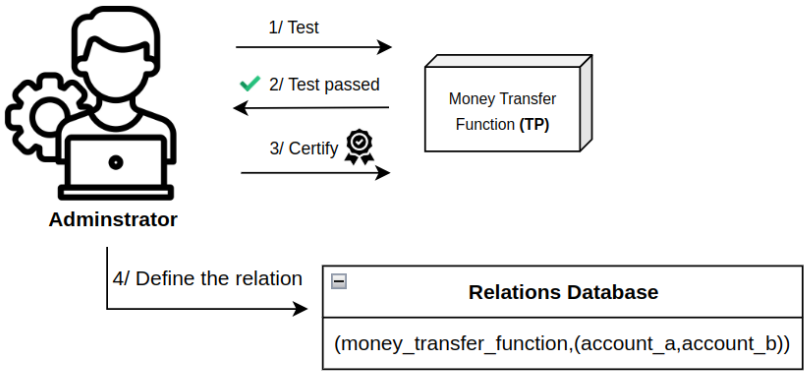


FIGURE 1.10: Money Transfer Function Certification Process

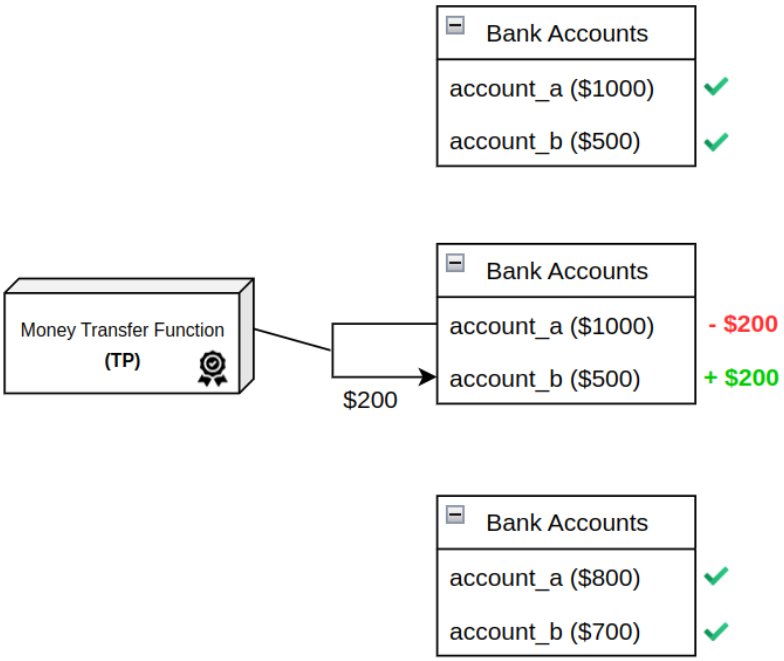


FIGURE 1.11: Money Transfer Process

- **C3 (Separation Of Duties)** The relations listed in E2 must be verified to comply with the separation of duties. [61]
  Example: Corporate Expense Approval System

  - **CDI:** Company expense report: $5,000 travel reimbursement
  - **TP:** Expense approval procedure
  - **Users:** Alice (Employee), Bob (Manager), Carol (Finance)

  Scenario: The system verifies that Alice (who submitted the expense) cannot also approve it or process the payment.

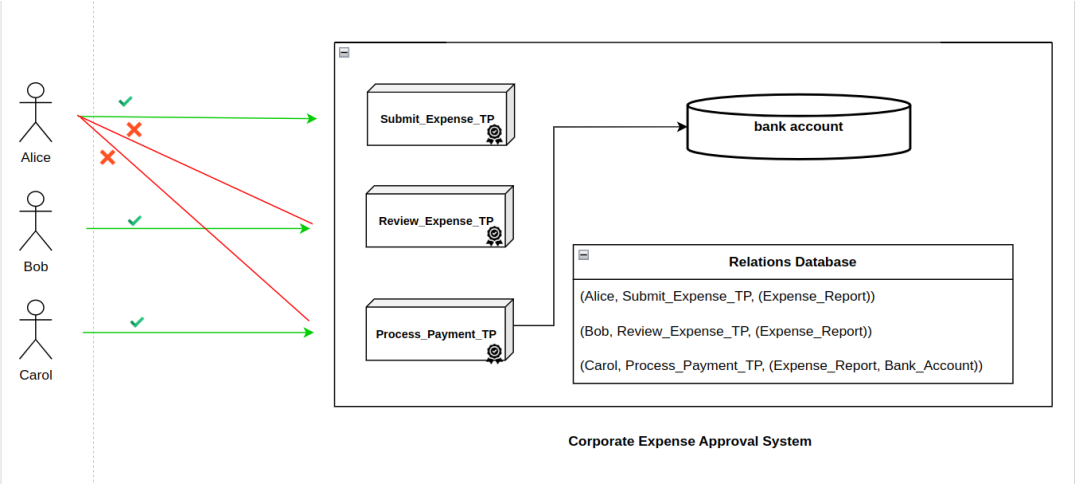


FIGURE 1.12: Separation Of Duties

- **C4 (Log Certification)** All TPs must be certified to ensure they generate log entries. [61]
  Example: ATM Cash Withdrawal System
  - **CDI:** Customer account balance: $2,000
  - **TP:** Cash_Withdrawal

  Scenario: Customer withdraws $100 from their account at an ATM.
  The `Cash_Withdrawal` TP has been certified to automatically generate comprehensive log entries for every withdrawal transaction.

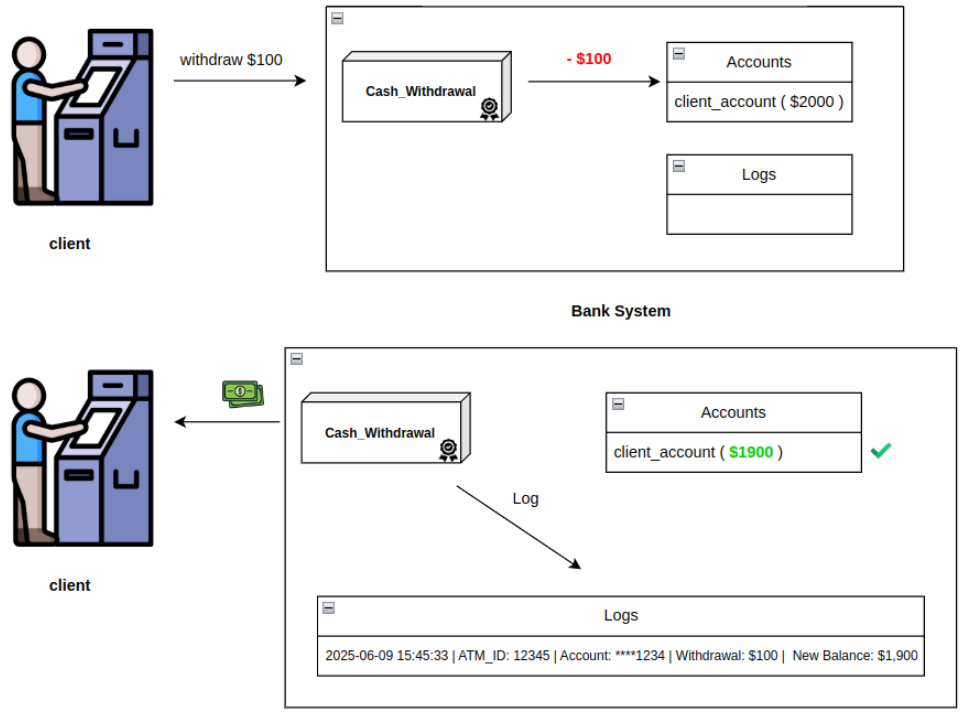


FIGURE 1.13: Logging

- **C5** TPs that transform UDIs into CDIs must be certified. [61]
  Example: Customer Registration System
  - **UDI**: Raw customer signup form data from website
    - Name: `"john smith"` (lowercase, no validation)
    - Email: `"johnsmith@gmail"` (incomplete format)
    - Phone: `"555.123.4567"` (mixed format)
  - **TP**: `Customer_Data_Validator` procedure
  - **CDI**: Clean, validated customer record in database

  Scenario: A new customer fills out an online registration form with messy, unvalidated data that needs to be converted into a proper database record. The Customer_Data_Validator TP has been certified to safely transform unreliable UDI input into trustworthy CDI format.

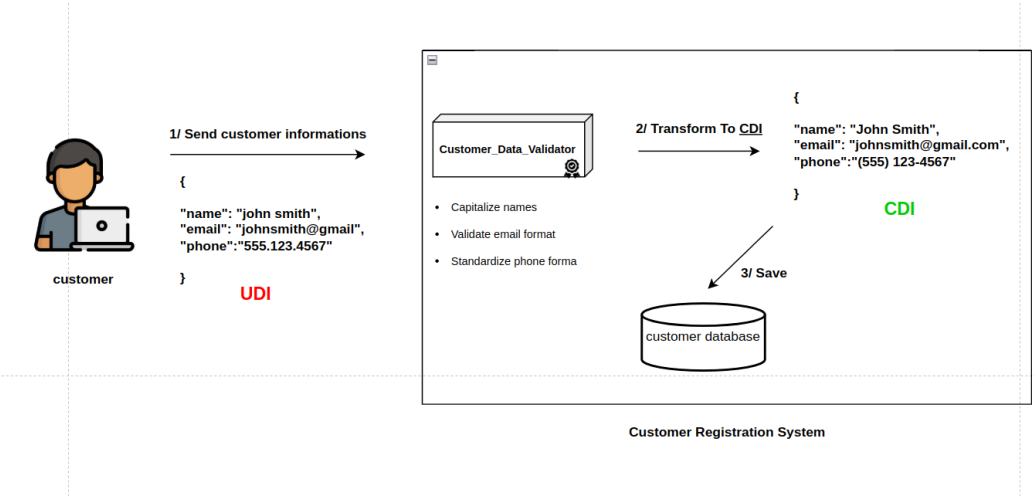


FIGURE 1.14: Transform UDI to CDI

1.4.3.2 Enforcement Rules

- **E1 (Execution of Validity)** The system must keep a list of C2 rule relations and ensure that CDIs are only changed through authorized TPs. [61]
  Example: Banking Account Balance System
  - **CDI:** Customer account balance: $5,000
  - **Authorized TPs:** Only three procedures are certified in the C2 relations list:
    - `ATM_Withdrawal_TP`
    - `Online_Transfer_TP`
    - `Bank_Deposit_TP`
  - **Unauthorized attempt:** Hacker tries to directly modify the balance database

  Scenario: Someone attempts to change the account balance to $50,000 by bypassing the authorized procedures and directly accessing the database.

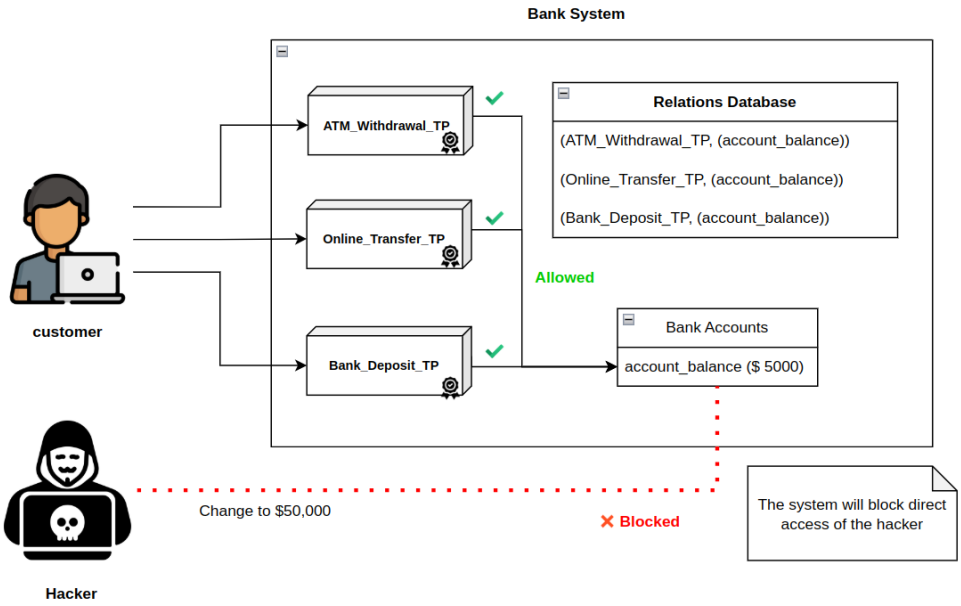


FIGURE 1.15: Change CDI only by authorized TPs

- **E2 (Execution of Obligations Separation)** The system must keep a list linking users, TPs, and the CDIs each TP can modify for a user. These relations look like: $(UserID, TP_i, (CDI_a, CDI_b, ...))$. [61]
  Example: University Grade Management System
  - **CDIs:** Student grade records for *"Database Systems"* course
  - **Users:** Professor Smith, TA Jennifer (Teaching Assistant), Registrar Mike
  - **TPs:**
    * `Enter_Grade_TP`
    * `Modify_Grade_TP`
    * `Finalize_Grade_TP`

  Scenario: TA Jennifer tries to modify an exam grade, but the system checks the E2 list and finds she's only authorized to enter quiz and homework grades.

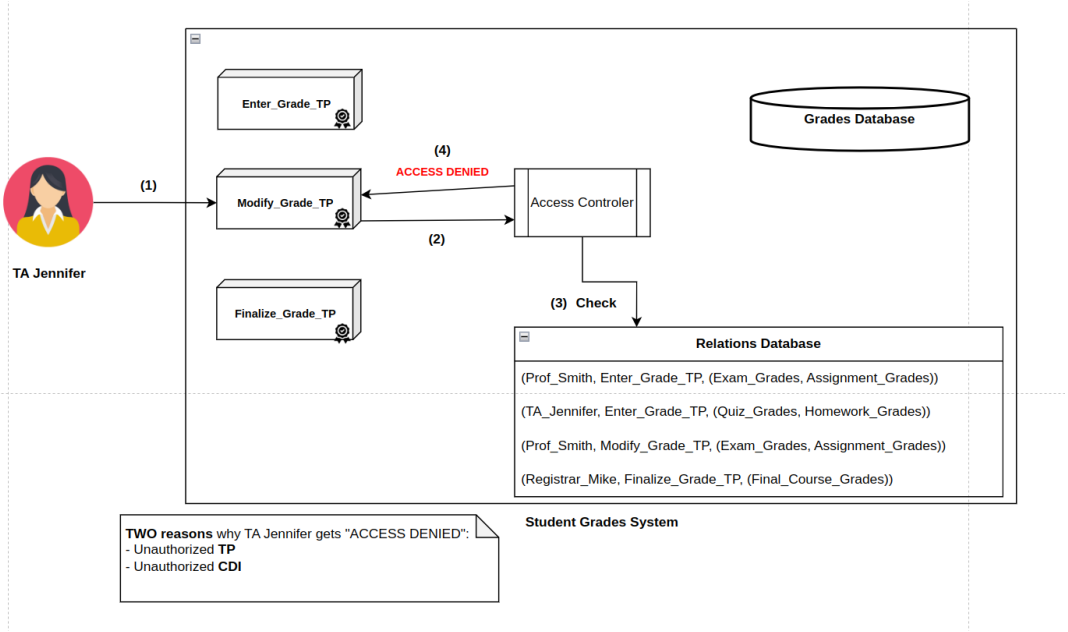


FIGURE 1.16: Applying User-TP-CDI relations

- **E3 (User Identity)** Users must be authenticated before using TPs. [61]
  Example: Online Banking System
  - **TP:** `Transfer_Money_TP` (moves funds between accounts)
  - **CDI:** Customer account balances
  - **User:** Sarah wants to transfer $500 to her friend

  Scenario: Sarah opens her banking app and immediately tries to initiate a money transfer without logging in.

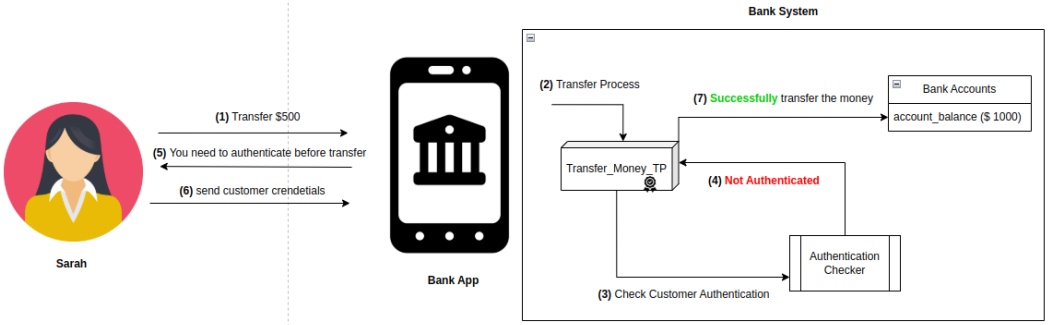


FIGURE 1.17: User Authentication

- **E4 (Initiation)** Only authorized users can define relations. [61]
  Example: Corporate IT Access Management System.
  - **Users:**
    - **Alice:** IT Security Administrator (*authorized*)
    - **Bob:** Regular Employee (*not authorized*)
    - **Carol:** Department Manager (*not authorized*)

  Scenario: Bob (regular employee) tries to create a new E2 relation to give himself access to the payroll system so he can modify his own salary record.

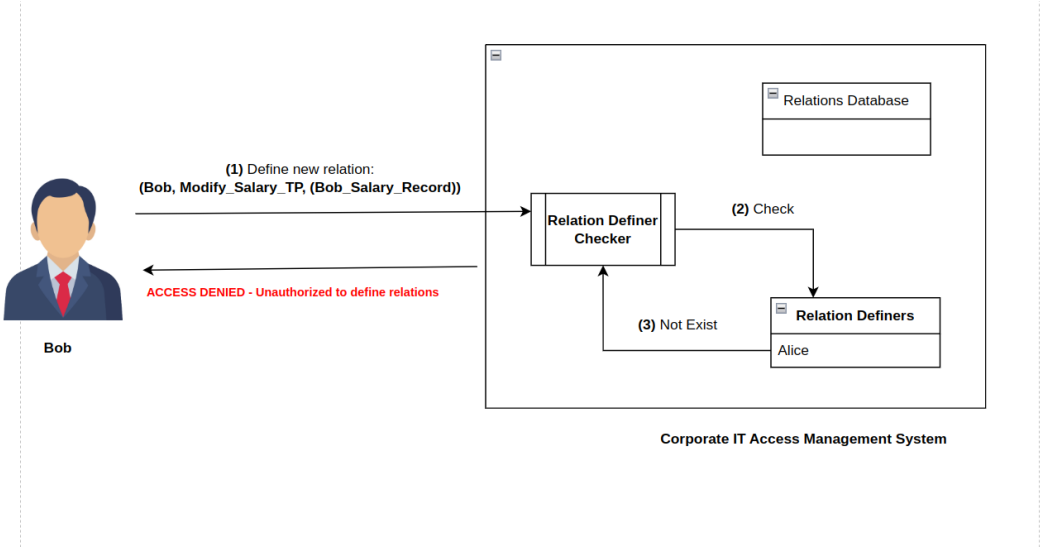


FIGURE 1.18: Bob is Unauthorized to define relations

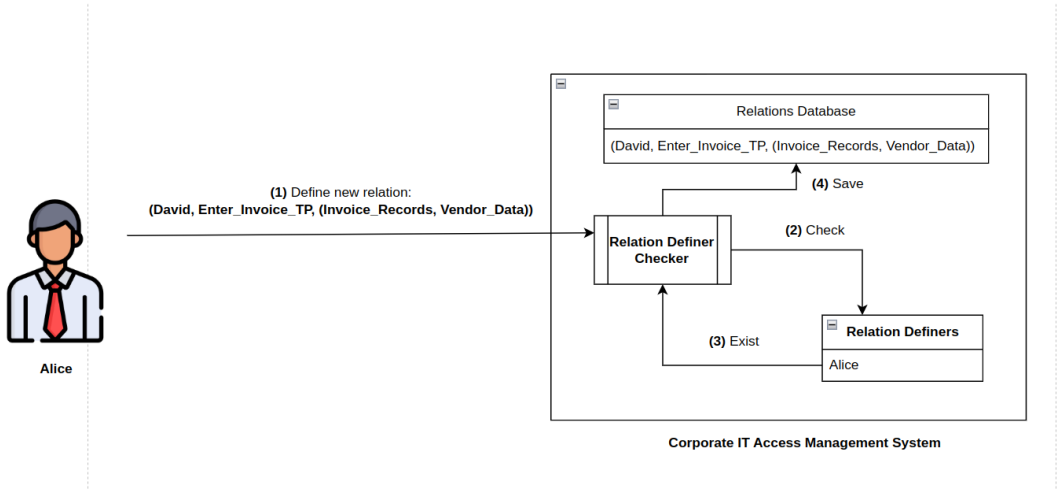


FIGURE 1.19: Alice is authorized to define relations

1.4.4 Clark Wilson Security Policy:

After defining the key principles and core concepts of the Clark-Wilson security Model, its elements, as well as its certification and enforcement rules, we have created this diagram 1.20 that outlines the scenario to be followed by the different entities and shows the application of each rule [21]:

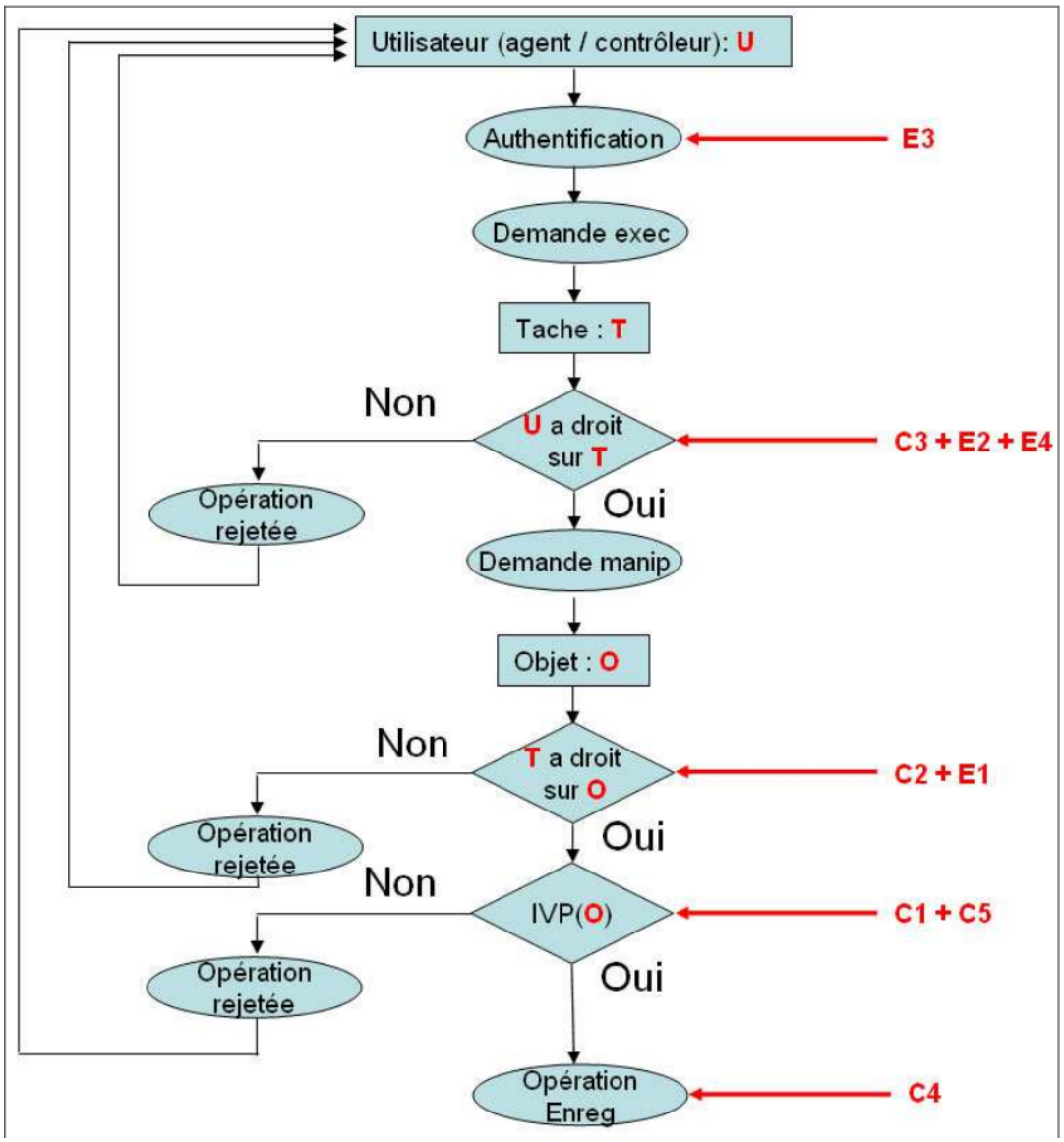


FIGURE 1.20: Clark Wilson Security Policy

1.5 Test-Driven Development

TDD is an agile software development methodology emphasizing writing automated tests before developing code. Like other agile practices, TDD promotes software development in small iterations. The TDD process starts by creating a failing test case for a specific functional requirement. Developers then write minimal production code to pass this test. After the test passes, the code is refactored to improve internal quality (e.g., readability, maintainability) without altering its external functional behavior. Unlike traditional development models that typically follow a linear sequence of design, coding, and testing, TDD inverts part of this sequence by introducing automated tests at the start of micro-iterations. TDD aims to reduce software defects and enhance code quality through early, continuous feedback [9, 53]. TDD typically uses automated unit testing frameworks (e.g., JUnit for Java, NUnit for .NET, CppUnit for C++) supporting frequent, consistent, and rapid test execution.
the figure 1.21 below summarizes the TDD cycle [28].

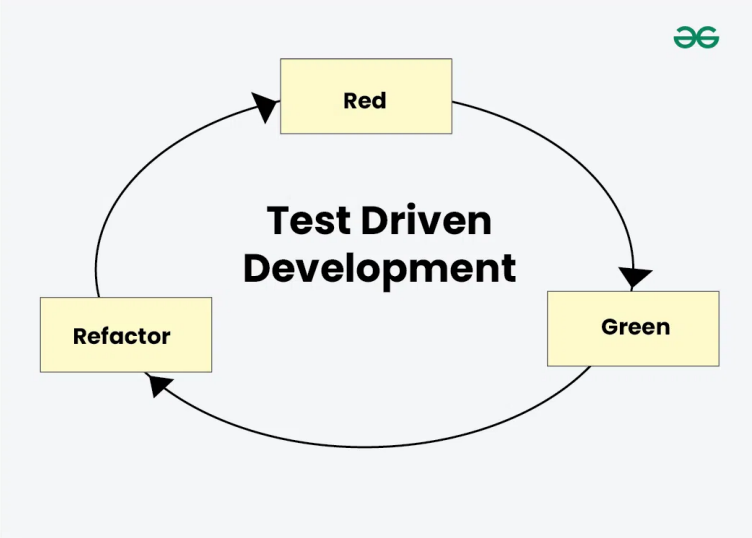


FIGURE 1.21: The Core Process of Test-Driven Development

Origin of TDD

Kent Beck formally introduced TDD in his 2003 book, *Test-Driven Development: By Example* [9]. However, writing tests before code existed earlier; NASA reportedly used a "test-first" approach during Project Mercury in the 1950s [24]. Beck adopted and refined this technique as a core practice within XP, where unit tests precede code implementation [10]. The success of this "test-first" approach in XP led to its formalization as TDD [9]. According to Beck [9], two rules define TDD:

1. Write new production code only when an automated test fails.
2. Eliminate duplication and improve design through refactoring once tests pass.

TDD is an iterative micro-process where tests derive from user requirements or functional specifications. Developers implement small functional increments to pass these tests, refactor the code, and repeat the cycle. Proponents argue this leads to higher code quality, fewer defects, and potentially lower long-term maintenance and testing costs, though empirical evidence varies [36, 42].

1.5.1 TDD Workflow

The TDD process, the "Red-Green-Refactor" cycle, involves five steps:

1. **Add a Test (Red):** Write an automated test for the next desired functionality based on requirements. It should fail initially as functionality is unimplemented.
2. **Run All Tests; See New One Fail:** Execute the test suite. The new test should fail (Red), verifying its correct formulation and that it doesn't pass without new code.
3. **Write Code (Green):** Implement minimal production code to pass the new test, focusing solely on passing, not perfect design.
4. **Run All Tests Again; See Them Pass:** Re-run the test suite. All tests, including the new one, should pass (Green), confirming the new code satisfies the test and hasn't broken existing functionality.
5. **Refactor Code:** With passing tests, improve the internal structure of new and related code (e.g., remove duplication, enhance clarity, improve design) without altering external behavior.

This cycle repeats for each new feature, driving development in small, validated steps. The figure 1.22 bellow summarizes the workflow

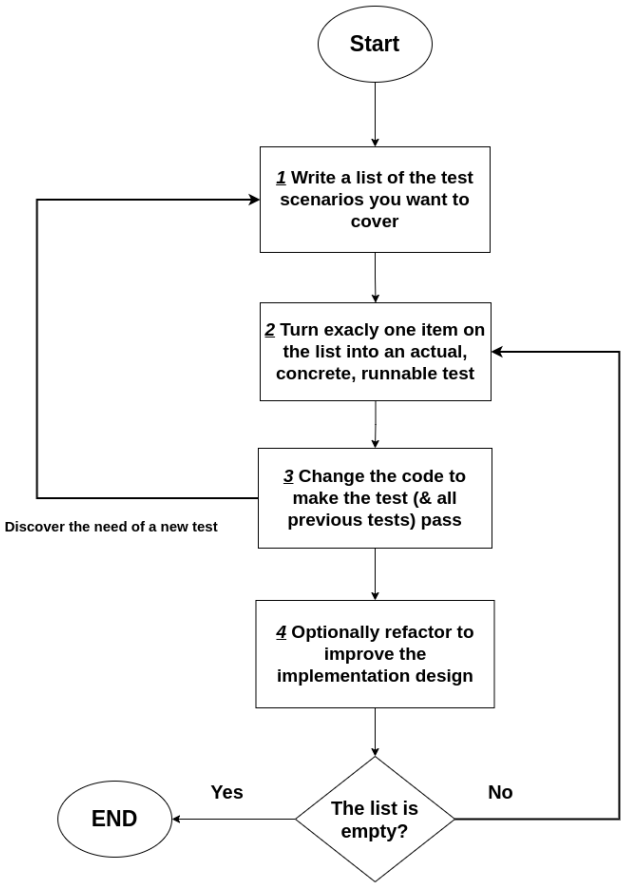


FIGURE 1.22: TDD Steps

1.5.2 Core Principles of TDD

TDD is founded on several core principles [9, 43]:

- **Test-First Development:** Writing tests before production code drives development. Developers use these tests to clarify requirements, define interfaces, and ensure initial code correctness.
- **Incremental Development and Feedback:** Software progresses in small, manageable increments, each validated by a test. This simplifies planning and execution, providing rapid feedback.
- **Frequent Testing and Continuous Integration:** Frequent automated test suite execution detects defects and regressions early, improving software quality and reducing risks during evolution.

1.5.3 Characteristics of Effective TDD Tests

TDD tests, primarily unit tests, differ from traditional testing artifacts. Effective TDD tests exhibit several characteristics [43, 45]:

- **Developer-Written:** Developers write them as they are integral to code design and implementation.
- **Focused and Fast:** Tests are concise, targeting a single functionality, and execute quickly to allow frequent runs without disrupting flow.
- **Readable and Maintainable:** Tests must be readable to serve as verification and living documentation. Maintainability is crucial as code evolves.
- **Independent and Isolated:** Tests run independently; one test's outcome must not affect others. This requires careful fixture and dependency management.
- **Deterministic (Repeatable):** Tests must produce the same result with the same input and state, ensuring reliable feedback.
- **Automated:** Tests execute automatically via a testing framework, minimizing error and enabling frequent execution.
- **Design-Informing:** Writing tests first often informs design, leading to modular, decoupled, testable architectures.

1.5.4 Key Benefits Attributed to TDD

TDD adoption offers several potential benefits [8, 9, 36, 41]:

- **Immediate Feedback and Safety Net:** Tests provide instant validation after small changes, highlighting design flaws or regressions early.
- **Early Defect Detection and Reduced Fix Costs:** Defects identified early (within minutes of coding) potentially reduce remediation cost and effort.
- **Clarified Requirements and Improved Design:** Writing tests first forces critical thought about requirements, outcomes, and edge cases, improving user need alignment and API design.

- **More Maintainable and Modular Code:** TDD promotes clean, loosely coupled designs with well-defined interfaces, which are easier to test. The test suite aids maintainability by ensuring changes don't introduce regressions.
- **Living Documentation:** TDD tests serve as executable documentation, demonstrating code behavior and aiding new developer onboarding.
- **Increased Confidence in Iteration and Refactoring:** Developers refactor or add features more confidently, knowing the test suite will likely detect unintended side effects.
- **Potentially Reduced Debugging Time:** A failing test typically localizes issues to recent, small code changes, speeding root-cause analysis compared to debugging larger, untested codebases.

Realizing these benefits depends on team experience, project context, and TDD discipline.

1.6 Related Works

Several studies have investigated integrating biometric technologies and the IoT in intelligent surveillance and access control systems:

- **Face Recognition Surveillance in Mining** [31] Hidayat et al. [31] introduced a facial recognition surveillance system to monitor personnel in mining operations, enhancing safety and regulating access in this high-security environment.
- **Biometric Fingerprint for Presence and Room Access** [47] Muslimin et al. [47] proposed a fingerprint-based biometric system to manage presence verification and regulate room access, demonstrating biometrics in physical access control. Figure 1.23 shows their system architecture.

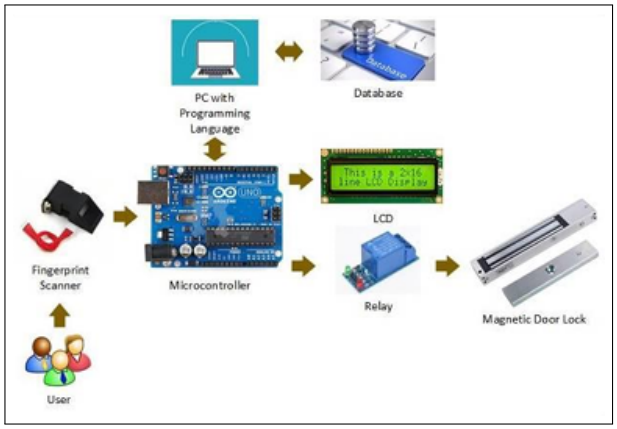


FIGURE 1.23: System architecture for fingerprint-based room access control

- **IoT System for a Smart University** [2] Afouf et al. [2] developed an IoT system for a Smart University, integrating connected devices to enhance academic environments through automation and intelligent control. Figure 1.24 illustrates their system.

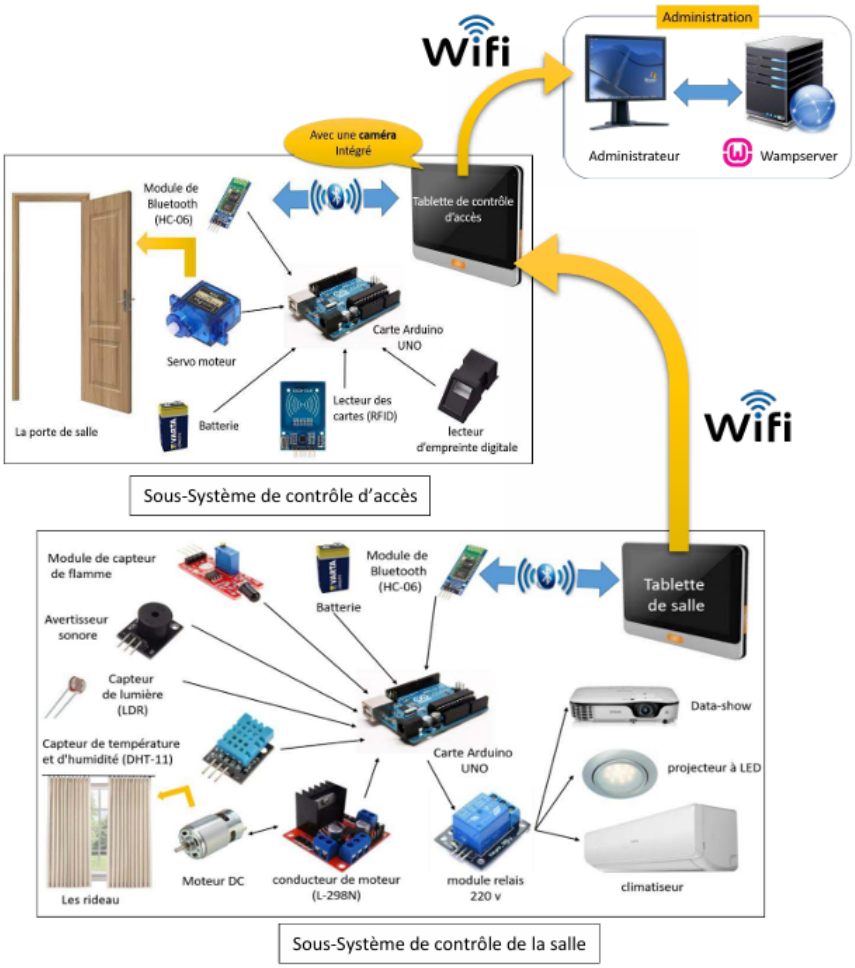


FIGURE 1.24: Smart university access control system

- **Design the smart security and surveillance systems:** System IN202311033719 [65] uses IoT sensors (ultrasonic, ESP32-CAM) and facial recognition for intrusion detection in smart cities. It supports real-time alerts, remote video monitoring, and power failure detection.

- **Smart Attendance via Fingerprint Recognition** [37] Kabir et al. [37] presented an automated fingerprint recognition system to streamline attendance tracking and leave management in academic environments. Figure 1.25 shows their system's flow diagram.

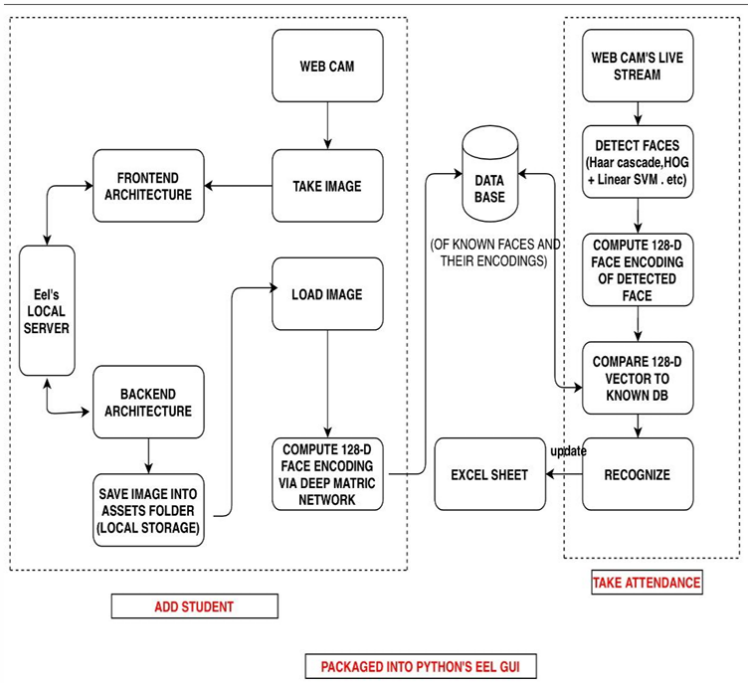


FIGURE 1.25: Flow diagram of the smart attendance system using fingerprint recognition

- **Smart CCTV cameras:**
  System IN202311052932 [66] features AI-based CCTV cameras that analyze video feeds in real time to detect unusual motion and send alerts. It lacks biometric verification and focuses mainly on visual surveillance.

- **Securing Smart Buildings with RFID and Fingerprints** Al et al. [5] investigated integrating RFID and fingerprint authentication to enhance smart building security, exploring combined token-based and biometric approaches. Figure 1.26 shows their system components.

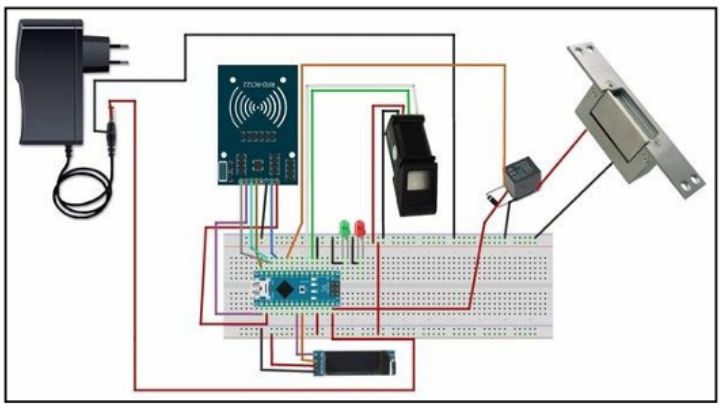


FIGURE 1.26: System components for securing smart buildings with RFID and fingerprint technologies

- **Smart surveillance for theft detection:**
  System IN202141000142 [64] uses PIR sensors and a Pi Cam with Raspberry

Pi to detect motion and identify faces using Haar Cascade. Alerts are sent via GSM after detecting unauthorized presence.

- **Smart surveillance system:**
  System IN202111010174 [63] combines smart cameras with PIR and sound sensors. It uses AI (TensorFlow) for object detection, encrypts communications, and stores data in the cloud. Cameras are activated only by suspicious motion or sound.

The following table 1.3 provides a comparative overview of these solutions, focusing on their main contributions and the limitations:

Reference	Year	Contribution	Limitations
[2]	2020	Proposed an IoT system for smart university labs integrating multiple access methods (QR code, RFID, facial, keypad, fingerprint) for lab automation.	Methods like QR and RFID only verify possession, not identity. No multi-factor authentication is implemented.
[47]	2021	Designed a fingerprint-based system for presence tracking and room access to improve building security.	Local fingerprint verification only. No central management or dual authentication, limiting identity assurance in large-scale environments.
[37]	2021	Developed a fingerprint-based smart attendance and leave management system for academic institutions.	Focused only on attendance. Lacks a formal security model to ensure integrity in broader access control environments.
[31]	2024	Introduced a face recognition-based surveillance system for mining industry safety and staff monitoring.	Uses only one biometric factor (face), without dual verification. No guarantee that the person wasn't replaced or spoofed.
[5]	2024	Combined RFID and fingerprint verification for smart building security, aiming to enhance access reliability.	Enhances authentication, but lacks a formal integrity model and real-time monitoring, making it vulnerable to violations.
[65]	2023	Proposed a smart surveillance system for cities using IoT devices (ESP32-CAM, sensors), facial recognition, and multi-zone intrusion detection.	Relies solely on face recognition without strict access control policies. No formal model like Clark-Wilson to ensure integrity or prevent identity spoofing.
[66]	2023	Presented an AI- and IoT-enabled CCTV solution for detecting suspicious movements in real time.	Visual surveillance only. No identity verification or secure access model. Lacks biometric checks or formal security enforcement.
[64]	2021	Designed a Raspberry Pi-based system combining motion detection and facial recognition for theft alerts.	Reacts after intrusion. Weak preventive mechanism. No identity verification before access. No guarantee of rightful user.
[63]	2021	Described a general-purpose surveillance system with AI-powered object detection, PIR sensors, and encrypted cloud storage.	Activated only upon detecting motion/sound. Lacks user verification before access. Does not ensure identity integrity.

Table 1.3: Comparison of Related Works

1.7 Conclusion

The integration of the IoT and AI is reshaping surveillance and access control systems, making them more intelligent, faster, and more responsive to dynamic security environments. Interconnected devices and sophisticated intelligent algorithms can collaborate to detect unusual/unauthorized behavior and initiate real-time responses. Ensuring these advanced systems' security and trustworthiness highlights the role of formal security models, particularly the Clark-Wilson integrity model. This model protects system data and control mechanism integrity, ensuring only authorized individuals access/modify sensitive information or execute critical operations. This formal security approach was lacking in many previous studies or conventional systems.

Integrating these technologies and security principles develops safer, more reliable surveillance and access control solutions for today's interconnected world. This highlights a critical gap in many current surveillance systems: the absence of enforced integrity policies. This thesis addresses this gap with its proposed solution.

The next chapter presents the design and implementation of a novel software library. This library builds on Clark-Wilson integrity model principles and was developed using a TDD approach for robustness and correctness.

Chapter 2

Design And Implementation of Clark Wilson Integrity Library

2.1 Introduction

This chapter outlines the development of a software library based on the Clark-Wilson integrity model.

We begin by discussing the library's design, explaining how it is structured into two distinct environments: the Execution Environment and the Decision Environment.

Next, we delve into the library's architecture, which is divided into two core components: the Policy Configuration Layer, responsible for managing the setup of security rules and relationships, and the Enforcement Layer, which ensures compliance during runtime operations. We provide use cases to illustrate how each layer contributes to maintaining data integrity in practical scenarios.

Finally, we walk through the TDD process, demonstrating how iterative testing guided the implementation of each module. Real-world examples highlight how writing tests first enabled us to build a robust, well-validated system for enforcing the integrity mechanisms defined by the Clark-Wilson model.

2.2 Clark-Wilson Library

The Clark-Wilson integrity model library is an implementation of the Clark-Wilson integrity model. The system has been divided into two distinct environments: the **Execution Environment** and the **Decision Environment**. It encompasses both policy configuration and runtime enforcement use cases, enabling structured management of security policies and their dynamic enforcement during system operation.

2.3 Library Design

1. **Execution Environment:** This environment is responsible for executing the use cases of the system that incorporates the library.

2. **Decision Environment:** This environment handles user authentication, determines whether a user is authorized to perform a specific action, verifies data integrity after a series of transactions, and manages logging.

The following figure 2.1 shows how the Clark-Wilson library is implemented in our system.

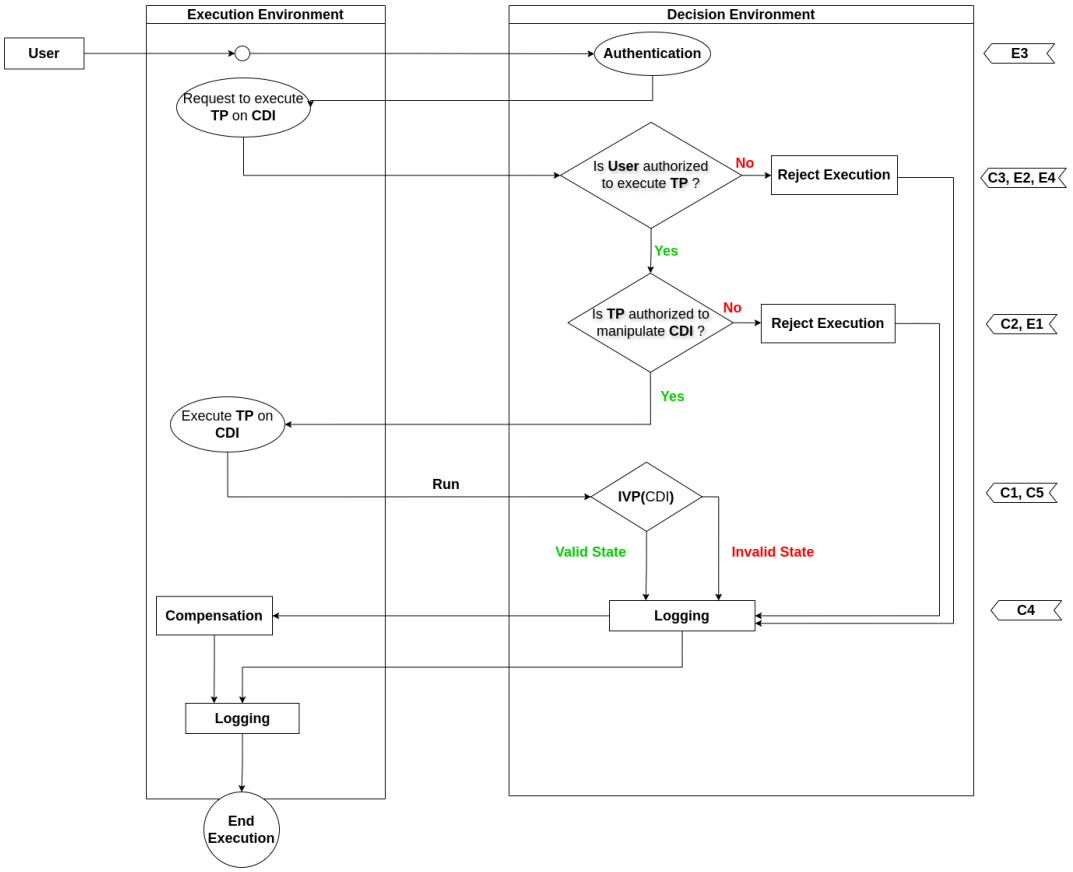


FIGURE 2.1: Clark-Wilson Library Design

2.4 Policy Configuration Use Cases

This layer is responsible for configuring the library for any software whose integrity we want to ensure. It establishes the security policy and relationships within the model.

Name	Description
Add TP CDI Certification	Creates relations by linking a `TP` to its authorized `CDI`, satisfying policy rule C2.
Assign Role TP	Links authorized **Roles** to a **TP**, implementing policy rule E2.
Add TP	Registers a new `TP` in the library and associates it with specific system functionality for handling `CDI`s and `UDI`s.
Add CDI	Defines a new `CDI` in the library and links it to a critical database element requiring integrity protection.
Add UDI	Registers a `UDI` in the library and associates it with a non-critical database element with fewer integrity constraints.
Add TP UDI Certification	Establishes certification relations authorizing specific `TP`s to modify designated `UDI`s, satisfying policy rule C5.
Authentication	Implements user authentication mechanisms required before invoking `TP`s, fulfilling policy rule E3.
Is Authorized Configuration	Controls access rights for defining new relations within the system, implementing policy rule E4.
Remove TP CDI Certification	Revokes existing certification relations between a `TP` and its authorized `CDI`s from the library's stored relations.
Remove TP UDI Certification	Deletes certification relations authorizing specific `TP`s to modify designated `UDI`s from the stored relations.
Remove TP	Removes a registered `TP` and all its associated certifications from the library.
Remove CDI	Deletes a stored `CDI` and removes all related certification associations from the library.
Remove UDI	Removes a registered `UDI` and all its associated certification relations from the library.
Remove Role TP Assignment	Revokes existing role-to-`TP` assignments from the stored authorization relations.
Edit TP CDI Certification	Modifies existing certification relations by updating the `TP` or `CDI` components within established relations.

Edit TP UDI Certification	Updates existing certification relations by changing the `TP` or `UDI` components within stored relations.
Edit TP	Modifies the system functionality associated with an existing `TP` while preserving its certification relations.
Edit CDI	Updates the database linkage associated with an existing `CDI` while maintaining its integrity constraints.
Edit UDI	Modifies the database association linked to an existing `UDI` while preserving existing certifications.
Edit Role TP Assignment	Updates existing role-to-`TP` assignments by modifying either the role or `TP` component within stored relations.

Table 2.1: Policy Configuration Use Cases

2.5 Runtime Enforcement Use Cases

Applications interact with runtime enforcement use cases at runtime to ensure data integrity per configured policies.

Name	**Description**
Is Authorized	Used by software to verify whether a requested action is authorized before execution.
IVPs Verifier	Executes Integrity Verification Procedures to confirm a **CDI** is in a valid state, satisfying policy rule C1.

Table 2.2: Runtime Enforcement Use Cases

2.6 Implementation

In the implementation, we applied **TDD** in conjunction with **AOP**

Remark. In this part I skipped writing minimal code, and I always write the whole code to make it easy to understand.

An example of a use case implementation: Add TP CDI Certification Use Case test cases

```python
def test_given_tp_and_cdi_when_use_case_is_executed_then_tp_cdi_certification_is_saved():

    # Arrange(Given)
    tp_cdi_dto = TPCDIDTO(
        tp_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
        cdi_id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
    )
    spy_tp_cdi_certification_repository = Mock(
        spec=TPCDICertificationRepositoryInterface
    )
    use_case = AddTPCDICertificationUseCase(tp_cdi_certification_repository)

    # Act(When)
    use_case.execute(tp_cdi_dto)

    # Assert(Then)
    expected_saved_certification = TPCDICertification(
        tp=UUID("550e8400-e29b-41d4-a716-446655440000"),
        cdi=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
    )
    assert spy_tp_cdi_certification_repository.save.assert_called_once_with(
        expected_saved_certification
    )
```

Listing 1: Add TP CDI Certification Use Case Unit Test

The test failed when executed.

Figure 2.2: Unit Test Failed

The production code written to make the test pass

Entities

```python
class ConstrainedDataItems(BaseModel):
    id: UUID
```

Listing 2: Constained Data Items Entity

```
class TransformationProcedure(BaseModel):
    id: UUID
```

Listing 3: Transformation Procedure Entity

```
class TPCDICertification(BaseModel):
    tp: TransformationProcedure
    cdi: ConstrainedDataItems
```

Listing 4: TP-CDI Certification Entity

Dependencies

```
class TPCDICertificationRepositoryInterface(metaclass=ABCMeta):
    @abstractmethod
    def save(self, tp_cdi_certification: TPCDICertification) -> None:
        """abstract method"""

```

Listing 5: TP CDI Certification Repository Interface

Values

```
class TPCDIDTO(BaseModel):
    tp_id: UUID
    cdi_id: UUID

```

Listing 6: TP-CDI Data Transfer Object (DTO)

Use Case Code

```
class AddTPCDICertificationUseCase:
    def __init__(
        self, tp_cdi_certification_repository: TPCDICertificationRepositoryInterface
    ):
        self.tp_cdi_certification_repository = tp_cdi_certification_repository

    def execute(self, tp_cdi_dto: TPCDIDTO):
        tp = TransformationProcedure(id=tp_cdi_dto.tp_id)
        cdi = ConstrainedDataItems(id=tp_cdi_dto.cdi_id)
        tp_cdi_certification = TPCDICertification(tp=tp, cdi=cdi)
        self.tp_cdi_certification_repository.save(tp_cdi_certification)

```

Listing 7: Add TP CDI Certification Use Case Production Code

After implementing the use case, the test passes.

FIGURE 2.3: Unit Test Passed

Refactoring

```
1  def test_given_tp_and_cdi_when_use_case_is_executed_then_tp_cdi_certification_is_saved():
2
3      def __assert_tp_cdi_certification_was_saved(_spy_repository):
4          expected_saved_certification = TPCDICertification(
5              tp=TransformationProcedure(id=UUID("550e8400-e29b-41d4-a716-446655440000")),
6              cdi=ConstrainedDataItems(id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da")),
7          )
8          _spy_repository.save.assert_called_once_with(expected_saved_certification)
9
10     # Arrange(Given)
11     tp_cdi_dto = TPCDIDTO(
12         tp_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
13         cdi_id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
14     )
15     spy_tp_cdi_certification_repository = Mock(
16         spec=TPCDICertificationRepositoryInterface
17     )
18     use_case = AddTPCDICertificationUseCase(spy_tp_cdi_certification_repository)
19
20     # Act(When)
21     use_case.execute(tp_cdi_dto)
22
23     # Assert(Then)
24     __assert_tp_cdi_certification_was_saved(spy_tp_cdi_certification_repository)
25
```

LISTING 8: Refactored Add TP CDI Certification Use Case Unit Test

An example of a use case implementation: TP CDI Certification Duplication Checker Aspect

```
def test_given_existing_tp_cdi_certification_when_use_case_is_executed_then_returns_failure_response():
    # Arrange(Given)
    tp_cdi_dto = TPCDIDTO(
        tp_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
        cdi_id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
    )
    dummy_use_case = Mock(spec=AddTPCDICertificationUseCase)
    tp_cdi_certification_existence_checker_repository = Mock(
        spec=AddTPCDICertificationExistenceCheckerRepositoryInterface
    )
    tp_cdi_certification_existence_checker_repository.exists_certification.return_value =
    (
        True
    )
    tp_cdi_certification_duplication_checker_aspect = (
        TPCDICertificationDuplicationCheckerAspect(
            tp_cdi_certification_existence_checker_repository
        )
    )
    aspect_list = [tp_cdi_certification_duplication_checker_aspect]
    decorated_use_case = DecoratedAddTPCDICertificationUseCase(
        dummy_use_case, aspect_list
    )

    # Act(When)
    actual_response = decorated_use_case.execute(tp_cdi_dto)

    # Assert(Then)
    expected_failure_response = AddTPCDICertificationUseCaseResponseDTO(
        status="Failed", duplicated_certification=True
    )

    assert actual_response == expected_failure_response

```

Listing 9: TP CDI Certification Duplication Checker Aspect Test Code

The test failed when executed.

FIGURE 2.4: TP CDI Certification Duplication Checker Aspect Test Failed

The aspect's production code written to make the test pass.

```
class TPCDICertificationDuplicationCheckerAspect(AspectInterface):
    def __init__(self, repository:
    ↪ TPCDICertificationExistenceCheckerRepositoryInterface):
        self.repository = repository
    def apply(self, _func: Callable[..., Any]) -> Any:
        @functools.wraps(_func)
        def wrapper(tp_cdi_dto: TPCDIDTO,*args, **kwarg) -> Any:
            def __certification_exists(tp_cdi_certification)->bool:
                return self.repository.exists_certification(tp_cdi_certific
                ↪ ation)

            tp = TransformationProcedure(id=tp_cdi_dto.tp_id)
            cdi = ConstrainedDataItems(id=tp_cdi_dto.cdi_id)
            tp_cdi_certification = TPCDICertification(tp=tp, cdi=cdi)

            if __certification_exists(tp_cdi_certification):
                return AddTPCDICertificationUseCaseResponseDTO(
                    status="Failure", duplicated_certification=True
                )
            return _func(tp_cdi_dto)

        return wrapper
```

LISTING 10: TP CDI Certification Duplication Checker Aspect Production Code

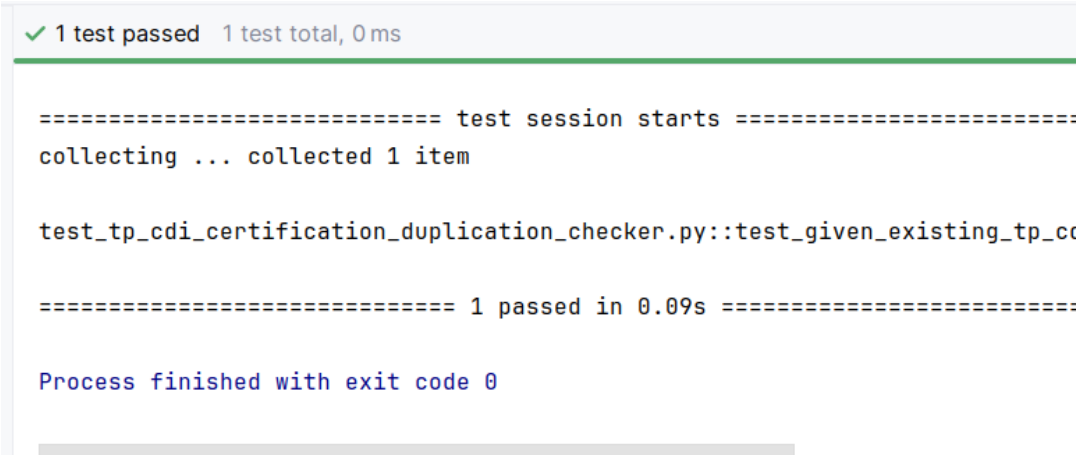


FIGURE 2.5: TP CDI Certification Duplication Checker Aspect Test Passed

```python
def test_given_existing_tp_cdi_certification_when_use_case_is_executed_then_returns_failure_response():
    def __assert_returns_failure_response(_actual_response):
        expected_failure_response = AddTPCDICertificationUseCaseResponseDTO(
            status="Failure", duplicated_certification=True
        )
        assert _actual_response == expected_failure_response

    # Arrange(Given)
    tp_cdi_dto = TPCDIDTO(
        tp_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
        cdi_id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
    )
    dummy_use_case = Mock(spec=AddTPCDICertificationUseCase)
    tp_cdi_certification_existence_checker_repository = (
        create_stub_tp_cdi_certification_existence_checker_repository(
            exists_certification=True
        )
    )
    decorated_use_case = build_decorated_add_tp_cdi_certification_use_case(
        dummy_use_case, tp_cdi_certification_existence_checker_repository
    )

    # Act(When)
    actual_response = decorated_use_case.execute(tp_cdi_dto)

    # Assert(Then)
    __assert_returns_failure_response(actual_response)


```

LISTING 11: TP CDI Certification Duplication Checker Aspect Refactored Test

```
1
2  def test_given_tp_cdi_certification_when_use_case_is_executed_then_returns_
   ↪  success_response():
3
4      # Arrange(Given)
5      tp_cdi_dto = TPCDIDTO(
6          tp_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
7          cdi_id=UUID("16fd2706-8baf-433b-82eb-8c7fada847da"),
8      )
9      in_memory_tp_cdi_certification_repository =
       ↪  InMemoryTPCDICertificationRepository()
10     in_memory_tp_cdi_certification_existence_checker_repository = (
11         InMemoryTPCDICertificationExistenceCheckerRepository()
12     )
13     decorated_use_case_builder =
       ↪  DecoratedAddTPCDICertificationUseCaseBuilder(
14         in_memory_tp_cdi_certification_repository,
15         in_memory_tp_cdi_certification_existence_checker_repository,
16     )
17     decorated_use_case = decorated_use_case_builder.build()
18
19     # Act(When)
20     actual_response = decorated_use_case.execute(tp_cdi_dto)
21
22     # Assert(Then)
23     expected_response = AddTPCDICertificationUseCaseResponseDTO(
24         status="Success", duplicated_certification=False
25     )
26     assert actual_response == expected_response
27
28
29
```

Listing 12: Add TP CDI Certification Use Case Acceptance Test

The acceptance test failed when executed.

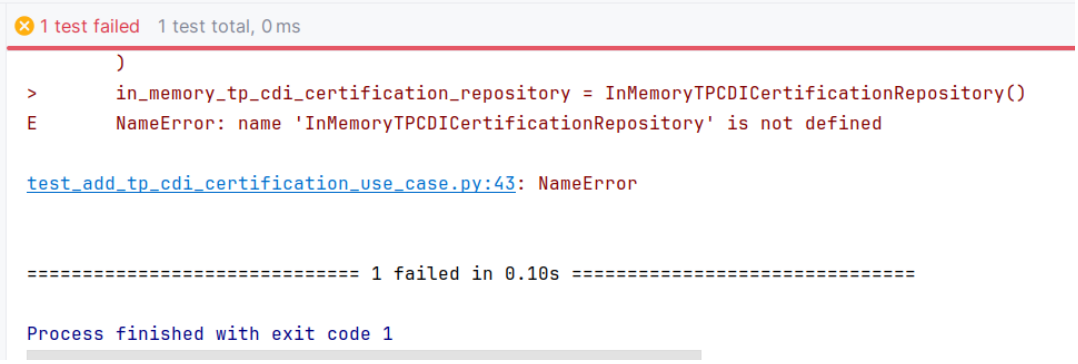


Figure 2.6: Add TP CDI Certification Use Case Acceptance Test Failed

The aspect's production code written to make the test pass.

```
1
2  class DecoratedAddTPCDICertificationUseCaseBuilder:
3      def __init__(
4          self,
5          tp_cdi_certification_repository:
           ↪ TPCDICertificationRepositoryInterface,
6          tp_cdi_certification_existence_checker_repository:
           ↪ TPCDICertificationExistenceCheckerRepositoryInterface,
7      ):
8          self.tp_cdi_certification_repository =
           ↪ tp_cdi_certification_repository
9          self.tp_cdi_certification_existence_checker_repository = (
10             tp_cdi_certification_existence_checker_repository
11         )
12
13     def build(self) -> DecoratedAddTPCDICertificationUseCase:
14         aspect_list = self.__build_aspect_list()
15         use_case = AddTPCDICertificationUseCase(self.tp_cdi_certification_r
           ↪ epository)
16         return DecoratedAddTPCDICertificationUseCase(use_case, aspect_list)
17
18     def __build_aspect_list(self) -> List[AspectInterface]:
19
20         tp_cdi_certification_duplication_checker = (
21             TPCDICertificationDuplicationCheckerAspect(
22                 self.tp_cdi_certification_existence_checker_repository
23             )
24         )
25         return [tp_cdi_certification_duplication_checker]
26
27
```

Listing 13: Decorated Add TP CDI Certification Use Case Builder

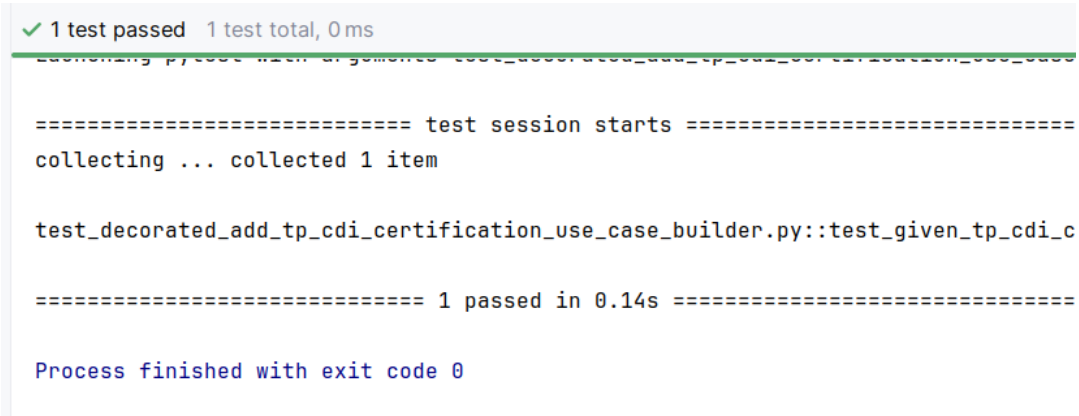


Figure 2.7: Add TP CDI Certification Use Case Acceptance Test Passed

```
class IVPsVerifierUseCase(IVPsVerifierUseCaseInterface):
    @staticmethod
    def execute(ivp_list: List[IVPInterface]) ->
    ↪ IVPsVerifierUseCaseResponseDTO:
        try:
            for ivp in ivp_list:
                ivp.verify()
            return IVPsVerifierUseCaseResponseDTO(state="Valid")
        except InvalidStateIVPException:
            return IVPsVerifierUseCaseResponseDTO(state="Invalid")


```

Listing 14: IVPs Verifier Use Case

2.7 Conclusion

This chapter detailed the design and implementation of a Clark Wilson integrity model library using **TDD**. The library was structured into two primary layers: **Policy Configuration Use Cases**, responsible for defining and managing security relationships, and **Runtime Enforcement Use Cases**, which ensure data integrity during system execution.

Each use case was carefully designed to enforce specific Clark Wilson policy rules, achieving full coverage of the model's integrity requirements. The rigorous application of TDD—supported by comprehensive testing at the unit, aspect, and acceptance levels—ensured that the implementation correctly enforces the intended security behaviors with a high degree of confidence.

Furthermore, the system was organized into two distinct environments:

1. **Execution Environment**: Handles the execution of use cases within the host system.

2. **Decision Environment**: Manages user authentication, authorization decisions, logging.

This separation enhances modularity and security by isolating critical decision-making processes from general operational logic.

The resulting library illustrates how formal security models can be transformed into practical, testable software components. By bridging the gap between theoretical security frameworks and real-world applications, this approach provides a reusable foundation for integrating Clark Wilson controls into diverse software systems and physical systems.

Chapter 3

Design And Implementation of Global System

3.1 Introduction

In this chapter, we describe the implementation details of the smart physical access control system.

The first section provides an overview of the overall physical and software architecture. In the physical architecture , we explain the internal subsystems and their respective roles. For the software architecture , we divided the entire system into two main components: the System Manager , responsible for managing users, roles, and monitoring, and the Door Controller , which handles access control for room doors.

In the second section, we present various scenarios that our system is designed to handle.

The third section includes comprehensive system diagrams for both the software and communication architecture.

In the fourth section, we dive deeper into the implementation details of both the System Manager and Door Controller , covering all aspects — physical, software, and communication. Throughout the implementation process, we followed the practices of the XP Agile framework , with a strong emphasis on TDD

While the system demonstrates a functional and modular design, several limitations should be acknowledged to contextualize its deployment potential and inform future improvements. First, biometric accuracy is affected by environmental conditions such as lighting and the quality of the fingerprint sensor. Second, the fingerprint module currently supports only 64 templates, which constrains its scalability in larger environments. Third, the limited processing power on edge devices restricts real-time detection performance. Additionally, fingerprints can be spoofed using materials such as gum or silicone, and the system remains vulnerable to spoofing through printed photographs or video replays. Finally, successful real-world deployment requires extensive field testing and the inclusion of hardware redundancy to ensure reliability.

3.2 System Overview

In this section, we begin by presenting a comprehensive overview of both the physical and software architectures, including their internal components and subsystems. We also provide explanatory figures to illustrate each part.

3.2.1 Physical Architecture

We divided our smart physical system into specific subsystems:

1. **Exterior Authentication Subsystem:** This includes dual biometric authentication (fingerprint and facial recognition) and is responsible for identifying the user to verify their authorization in the Door Controller Subsystem. 1, 2 in the figure 3.1.

2. **Open Door Subsystem:** This handles unlocking the door when the user presses the open button. 3, 4 in the figure 3.1.

3. **Interior Authentication Subsystem:** Also equipped with dual biometric authentication (fingerprint and facial recognition), this subsystem performs integrity verification after entry to ensure the correct user has accessed the space. 5, 6 in the figure 3.1.

4. **People Count Subsystem:** This tracks the number of people inside the room before and after entry, helping to maintain count integrity for security. 7 in the figure 3.1.

5. **Alarm Subsystem:** This subsystem is triggered when either the people count integrity check or the authentication integrity verification fails, activating an alarm to alert administrators of a potential security issue. 8, 9, 10 in the figure 3.1.

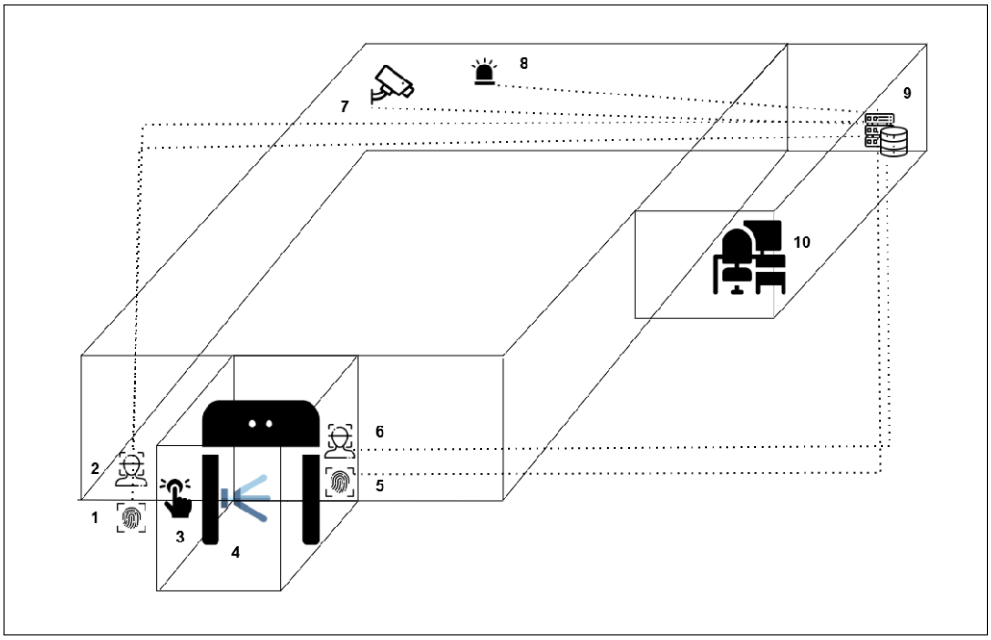


FIGURE 3.1: Physical System Architecture

3.2.2 Software Architecture

On the software side of our smart physical access control system, we have two internal systems:

1. **System Manager:** Responsible for managing users and their roles (such as adding users, creating roles, and assigning users to roles), as well as monitoring system activity.

2. **Door Controller System:** Handles user authentication to determine access permission, verifies access integrity, controls the door lock, and triggers the alarm if unauthorized access is detected.

Figure 3.2 illustrates the high-level architecture of the software and how its components interact

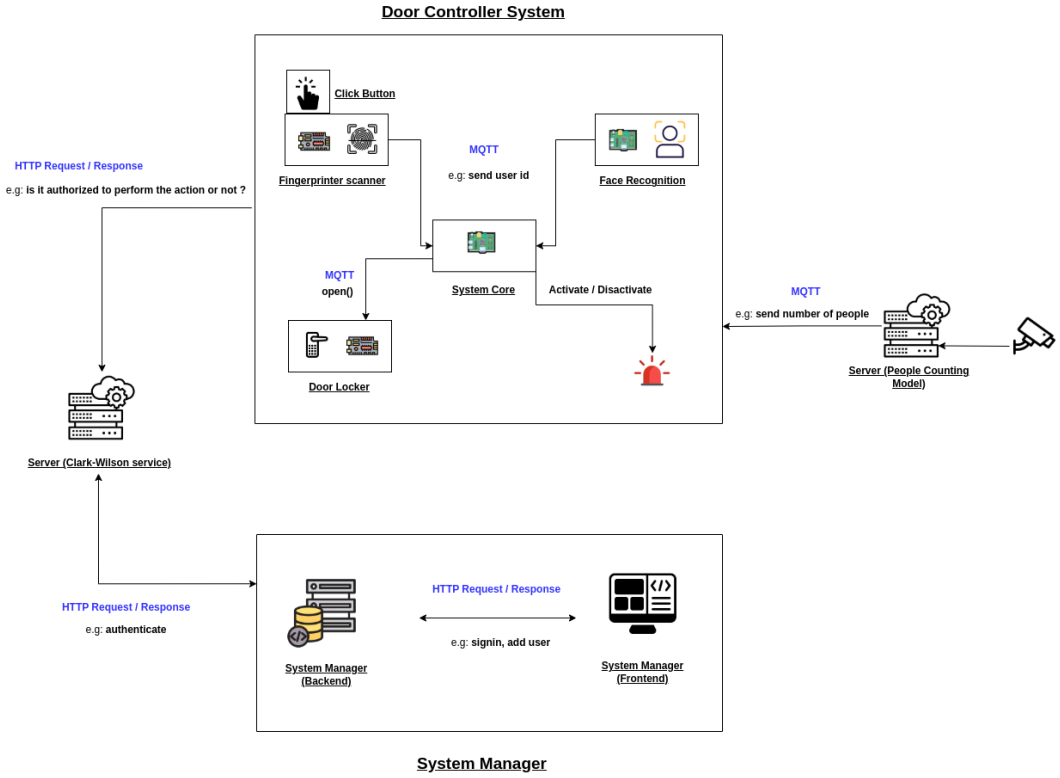


FIGURE 3.2: High-level Architecture Diagram of the Intelligent Surveillance System.

3.3 System Scenarios and Behavior

After we present an overview about our system. In this section, w'll outline four critical scenarios to demonstrate the operational integrity, fault detection, and security enforcement of the intelligent access control system. Each case illustrates how the combination of biometric authentication, real-time analytics, and the Clark-Wilson model ensures robust access control in a sensitive facility.

3.3.1 Scenario 1: Unauthorized Person Attempts Entry

the figure 3.3 summarize the scenario.

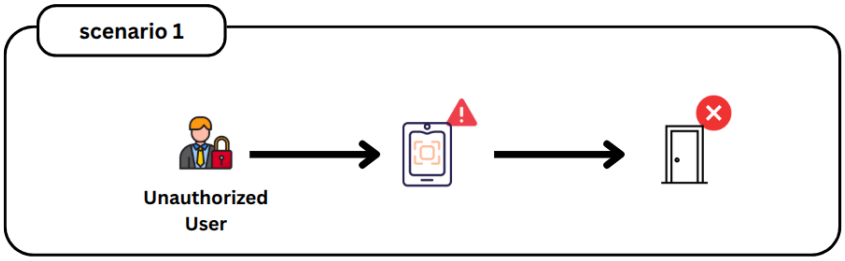


FIGURE 3.3: Unauthorized Person Attempts Entry.

3.3.1.1 Description

An individual who is **not registered** in the biometric database attempts to access the secure facility by presenting themselves at the door's biometric scanner.

3.3.1.2 System Behavior

- The facial recognition and fingerprint modules independently attempt to identify the individual.
- The DeepFace module fails to match the face with any entry in the authorized user database.
- The Clark-Wilson model, acting as a gatekeeper via its certified TPs, rejects the access attempt since no valid authentication was established.
- The door remains locked.
- An MQTT message is triggered to the backend, logging the failed attempt with timestamp and camera snapshot.
- A notification is optionally sent to security personnel.
- The real-time web interface displays a red warning and logs the identity spoofing attempt.

3.3.1.3 Outcome

Unauthorized access is denied, and the system maintains integrity by blocking all uncontrolled data/state changes.

3.3.2 Scenario 2: Authorized Person Opens Door — But Intruder Enters

the figure 3.4 summarize the scenario.

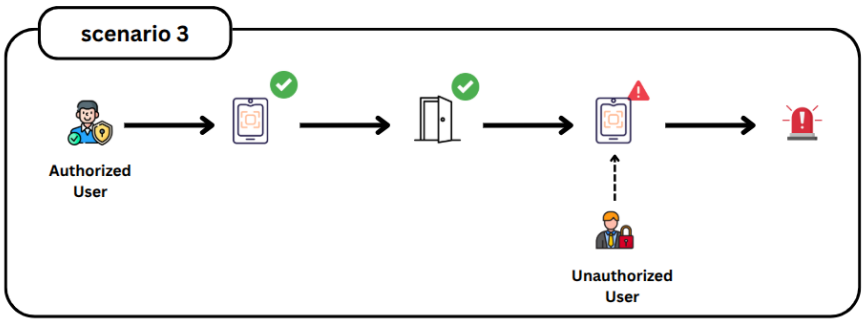


FIGURE 3.4: Authorized Person Opens Door — But Intruder Enters.

3.3.2.1 Description

An authorized user successfully authenticates at the outdoor biometric checkpoint (face + fingerprint). However, for some reason (e.g., distraction or trap), the authorized person does not enter. Instead, an unauthorized individual seizes the opportunity, passes through the open door alone, and proceeds toward the indoor biometric checkpoint to fake identity or gain access.

3.3.2.2 System Behavior

- Outdoor biometric (DeepFace + fingerprint) verifies the authorized person.
- The Clark-Wilson integrity model confirms this is a valid transformation request and allows the door to unlock.
- The YOLO-based people counting module detects only one person entering, which aligns with expectations.
- However, that person is not the one who passed the outdoor biometric check.
- At the indoor biometric checkpoint:
  - The intruder attempts to verify identity.
  - Their face and fingerprint do not match the identity that was authorized moments ago.
- The system performs cross-verification:
  - **ID used to open the door $\neq$ ID of the person trying to verify inside.**.

- Clark-Wilson model detects a violation of subject-object integrity:
  - The person attempting to reach the secure zone has not gone through a certified Transformation Procedure (the original outdoor biometric verification was for someone else).
- Immediate actions triggered:
  - Access is denied.
  - Alarm is triggered.
  - MQTT message is published with:
    - Identity used to unlock door
    - Identity mismatch detected at indoor checkpoint
    - Image/video of the intruder
    - Entry time
  - Web interface shows the discrepancy and flags a security breach.
  - All internal access is locked until resolved by authorized security personnel.

3.3.2.3 Outcome

Even though YOLO people counting does not detect excess entry (1 expected, 1 detected), the indoor biometric checkpoint enforces identity continuity and detects the substitution attack. The Clark-Wilson model blocks the transformation attempt and triggers active defense mechanisms.

3.3.3 Scenario 3: Multi-Person Entry Fraud (Piggybacking or Group Entry)

the figure 3.5 summarize the scenario.

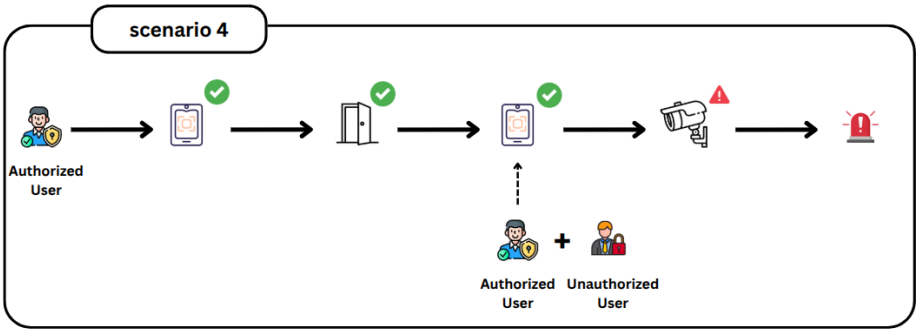


FIGURE 3.5: Multi-Person Entry Fraud (Piggybacking or Group Entry).

3.3.3.1 Description

An authorized user successfully completes biometric verification at the outdoor checkpoint. However, multiple individuals (authorized or not) enter the facility together — attempting to bypass identity verification by taking advantage of the open door.

3.3.3.2 System Behavior

- Outdoor biometric verification (facial and fingerprint) approves a single authorized user.
- The door unlocks, and individuals proceed inside.
- Indoor biometric scanner prompts for mandatory re-verification.
- Only one person verifies, claiming the previously authenticated identity.
- After indoor verification, the YOLO-based people counting module activates and detects that more than one person passed through the door.
- The number of individuals detected exceeds the number of verified users (only one was verified).
- The Clark-Wilson model detects a violation: multiple subjects attempting to access a secure area with a single valid transformation procedure (biometric pass).
- The system immediately:
  - Logs the violation.
  - Triggers a security alarm.
  - Captures and stores camera footage as forensic evidence.
  - Sends an MQTT alert with:
    - Number of detected individuals.
    - ID of the authenticated user.
    - Indoor video capture/image.
  - Smart lock re-engages to seal access.
  - All internal access systems are frozen until manually reset by a security officer.

3.3.3.3 Outcome

The system flags the piggybacking or group entry attempt after biometric re-verification, ensuring identity consistency and individual-only entry enforcement. The Clark-Wilson integrity model successfully blocks the state transition for non-verified individuals, while the incident is logged for audit and review.

3.3.4 Scenario 4: Legitimate Authorized Access

the figure 3.6 summarize the scenario.

3.3.4.1 Description

An authorized individual approaches the access point, undergoes biometric verification, and enters the room alone as per standard protocol.

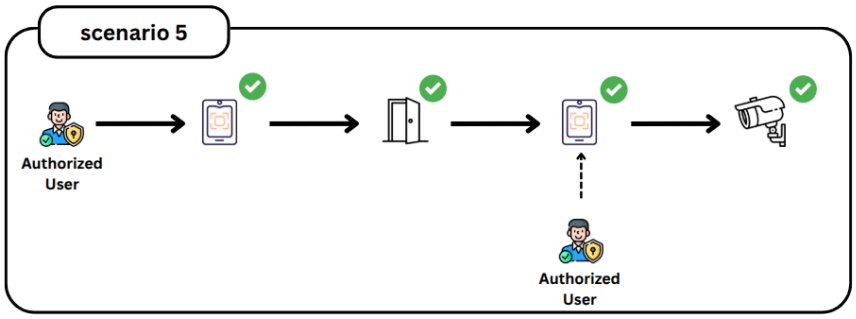


FIGURE 3.6: Legitimate Authorized Access.

3.3.4.2 System Behavior

- Dual biometric authentication (**face + fingerprint**) is successfully completed.
- The Clark-Wilson model confirms that this is a valid, well-formed transaction.
- Door unlocks, and the person enters.
- YOLO module detects exactly one person crossing the threshold.
- Internal checkpoint may optionally re-authenticate using face recognition to prevent impersonation inside.
- MQTT communication ensures all devices log the event:
  - Access time
  - Person ID
  - Camera snapshot
- Real-time web interface updates the entry log and internal state.
- The door automatically closes and locks after entry is complete.

3.3.4.3 Outcome

Smooth and secure access, without incident. System integrity is preserved and audit logs are updated.

Access Control Scenarios Summary Table

Scenario	Outdoor Biometric	Indoor Biometric	YOLO People Count	Clark-Wilson Check	Alarm	System Response
Unauthorized Person Attempts Entry	Rejected	—	0	No transformation allowed	No	Door remains locked, access denied
Authorized Person Opens Door, Intruder Enters	Authorized person	Intruder fails or mismatch	1 (Intruder alone)	Identity mismatch violation	Yes	Intruder blocked, breach logged, alarm triggered
Multi-Person Entry Fraud	One person verified	One person re-verified	more than 1	Count exceeds verified users	Yes	Access blocked, alarm triggered, lock re-engaged
Legitimate Authorized Access	Face + Fingerprint	Same person	1	Valid transformation	No	Door opens, access granted, entry logged

TABLE 3.1: Summary of Access Control Scenarios

3.4 System Design

As mentioned in the Introduction, throughout the development cycle of the entire system and its subsystems, we followed XP. This methodology doesn't emphasize drawing detailed diagrams. Instead, it prioritizes writing clean, expressive, and functional code over extensive upfront documentation or modeling. As a result, the diagrams presented in this section are used for illustration and communication with readers, rather than for formal documentation.

3.4.1 Software Design

We provide different types of diagrams for both the System Manager and the Door Controller System to help the reader gain a deeper understanding of our work.

3.4.1.1 System Manager Design

The System Manager plays a central role in orchestrating the overall functionality of the access control system. This subsection provides a detailed design overview of the System Manager, focusing on its use cases, component architecture, and interaction flows between the frontend and backend components. The following diagrams illustrate key aspects of the System Manager's design:

System Manager Use Case Diagram

To illustrate the roles and interactions within the system, Figure 3.7 provides the use case diagram of the System Manager.

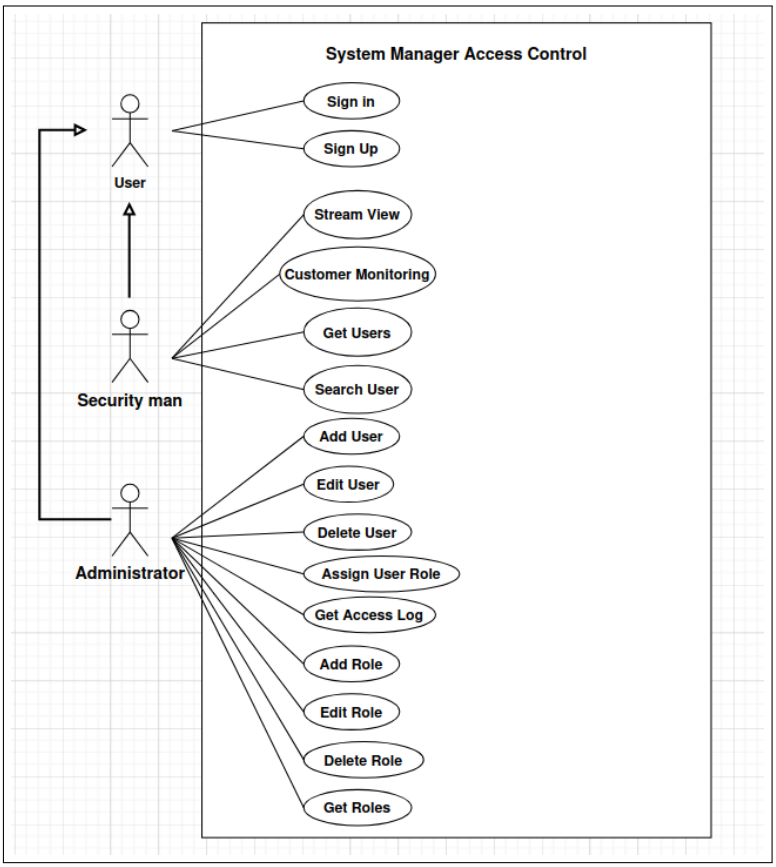


FIGURE 3.7: System Manager Use Case Diagram

System Manager Component Diagram

The component-level architecture of the System Manager is depicted in Figure 3.8, highlighting its internal modules and external interactions.

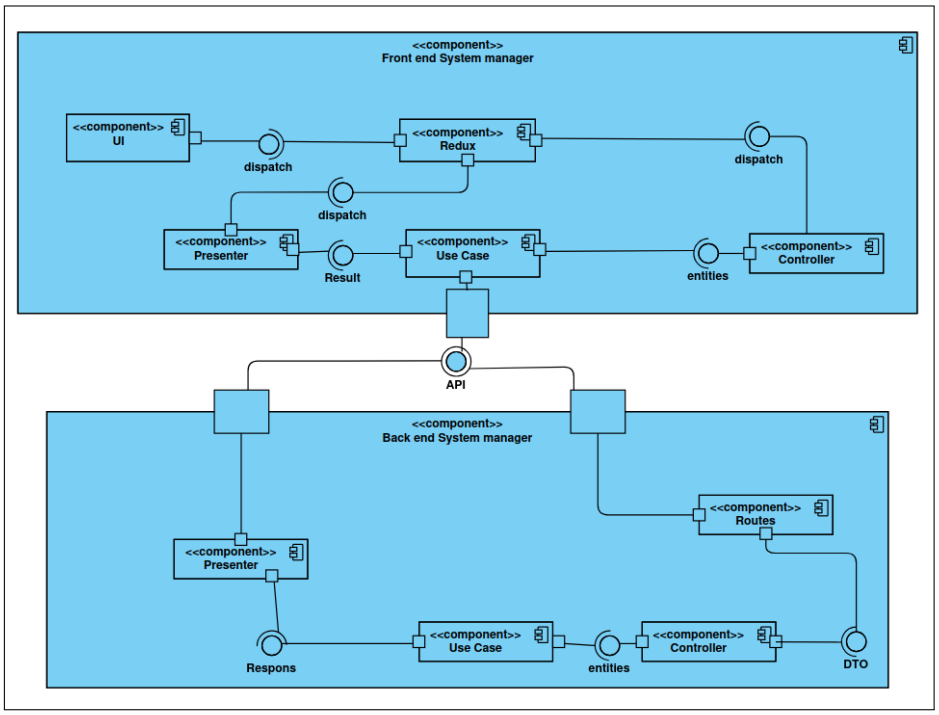


FIGURE 3.8: System Manager Component Diagram

Door Controller System Design

The component diagram of the Door Controller System is shown in Figure 3.9. It outlines the execution and decision environments, including the subcomponents and their communication interfaces.

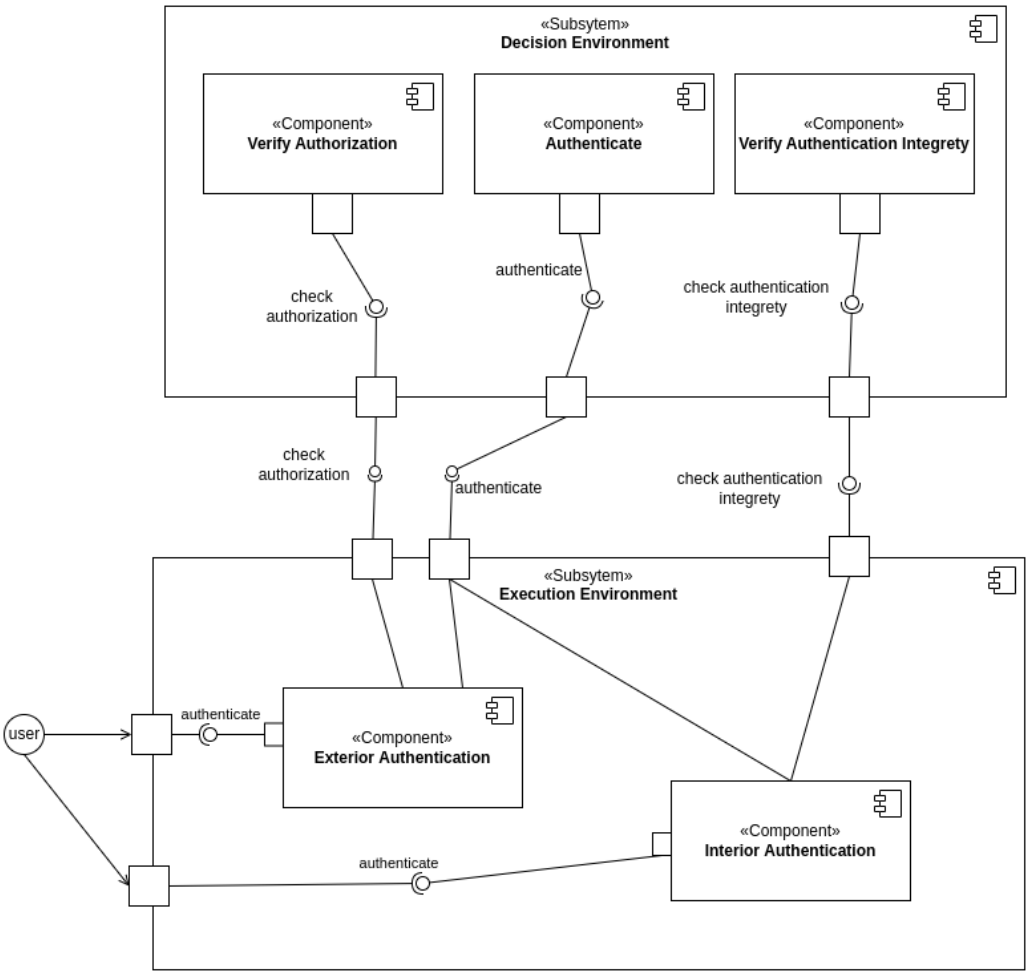


FIGURE 3.9: Door Controller System Component Diagram

Frontend Sequence Diagram Sign In

The frontend interaction for the sign-in process is detailed in Figure 3.10 through a sequence diagram.

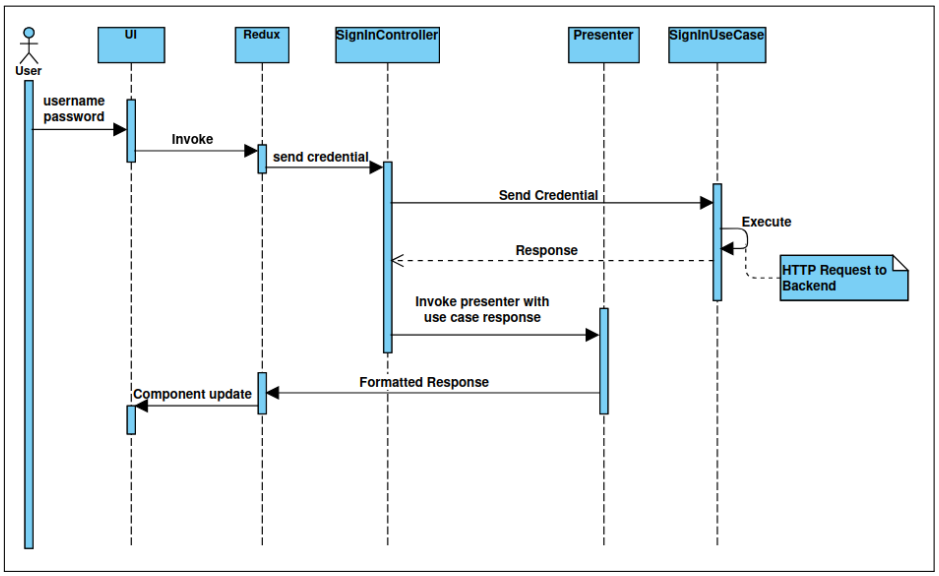


FIGURE 3.10: Sequence Diagram Sign In Frontend

Correspondingly, the backend processing flow for the same use case is illustrated in Figure 3.11.

Backend Sequence Diagram Sign In

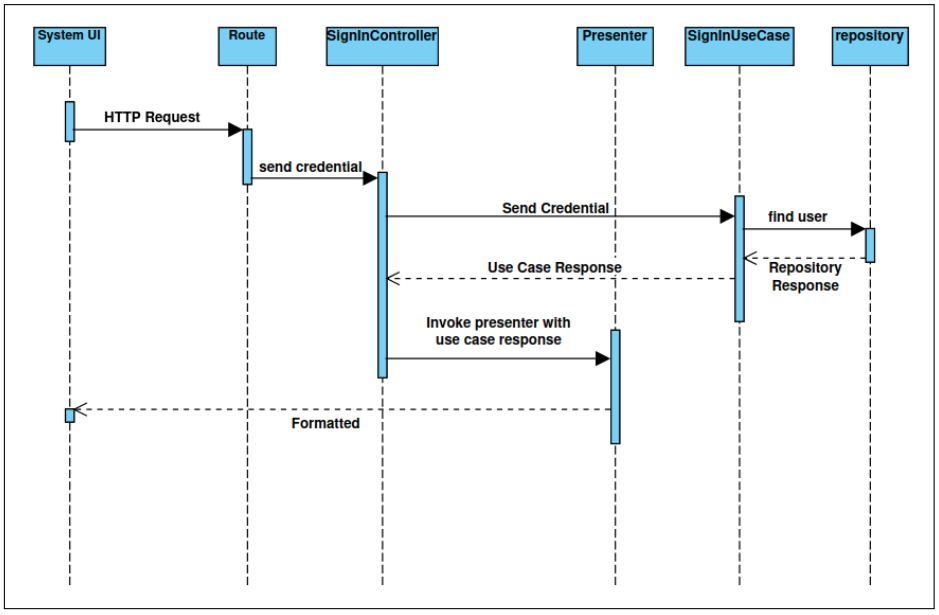


FIGURE 3.11: Sequence Diagram Sign In Backend

Sign In Class Diagram

The Figure 3.12 show some our **Modern AOP**

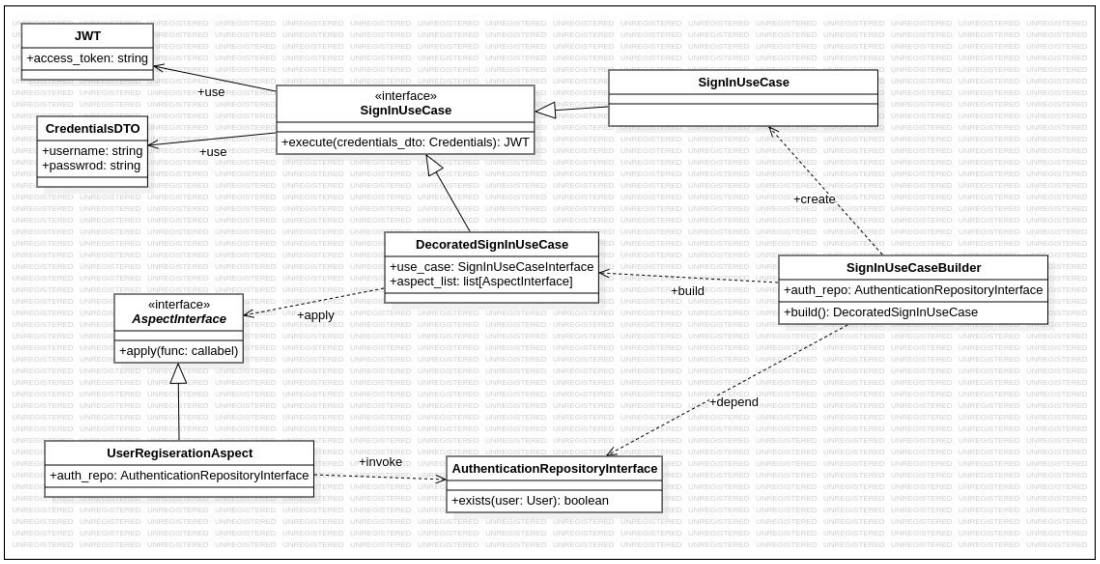


FIGURE 3.12: SignIn Class Diagram

3.4.2 Communication Design

The communication design of the system, as illustrated in Figure 3.13, showcases the interactions and data flow among the main components responsible for access control. The system's *core logic* acts as the central processing unit that receives inputs from various modules such as the *fingerprint recognition system*, *face recognition module*, *people counter*, and *manual press button*. These components send specific identifiers or signals—like `sendFingerprintID`, `sendFaceID`, and `sendPeopleCount`—to the core logic, which then makes decisions such as opening the lock or triggering alerts. For instance, if a valid identity is confirmed and the people count matches policy requirements, the core logic will issue an `open` command to the locking mechanism. Additionally, if the manual button is pressed or if an anomaly is detected, an `active` signal is sent to the alert system to take appropriate action. This well-structured communication architecture ensures secure, accurate, and efficient operation of the smart access system.

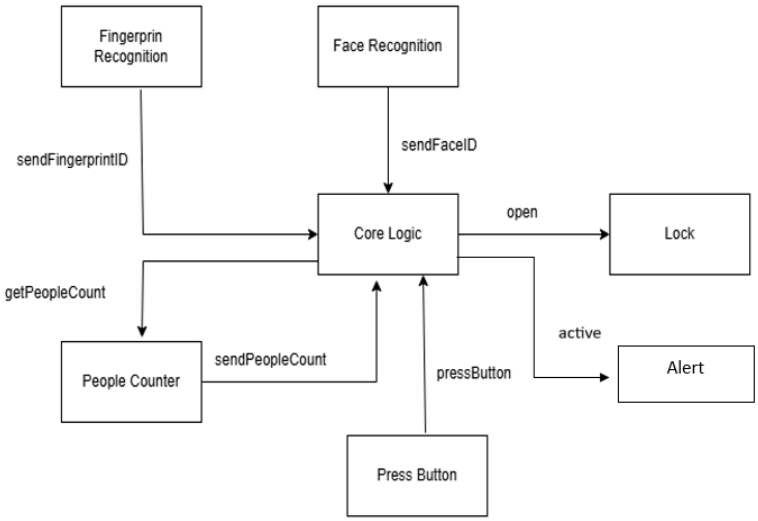


FIGURE 3.13: Network Communication Diagram.

3.5 System Implementation

As we described the system design in the last section. This section presents the implementation details of the system, covering both the hardware and software aspects. The system was developed using a modular approach, allowing for flexibility, scalability, and ease of maintenance. The implementation is divided into three key areas: the technology stack, physical subsystem integration, and software communication mechanisms.

Technology Stack:

- **IDEs:** PyCharm Professional (for Python), WebStorm (for JavaScript/TypeScript), and Arduino IDE (for microcontroller programming).
- **React with TypeScript:** Used to develop type-safe, scalable, and reusable user interface components.
- **Redux Toolkit:** Provides centralized state management to ensure predictable and maintainable application state across the frontend.
- **TailwindCSS:** A utility-first CSS framework that enables rapid, responsive UI development without the need to write extensive custom styles.
- **Jest and React Testing Library:** Utilized for writing unit and integration tests that simulate real user interactions, ensuring frontend reliability.
- **Vite:** A modern front-end build tool known for its fast startup time and near-instant hot module replacement (HMR) during development.
- **Axios:** A promise-based HTTP client for making secure and well-structured API requests to backend services.
- **Python:** Selected as the primary backend language for its simplicity, flexibility, and rich ecosystem for data handling and scripting tasks.

- **FastAPI:** A high-performance Python web framework used to build RESTful APIs, offering asynchronous support and automatic OpenAPI documentation generation.
- **Pytest:** A robust testing framework for writing clean, scalable unit and integration tests in the backend.
- **paho-mqtt:** A lightweight MQTT client library that facilitates communication with IoT devices using a publish-subscribe messaging pattern.

We adopted a TDD workflow throughout the project lifecycle. This approach enabled early detection of bugs, improved code architecture, and provided greater confidence during refactoring efforts, as discussed in Section 1.5.

3.5.1 Physical Implementation

Here we will describe how the access control system is physically assembled. We describe the real hardware components used, how they are connected, and how each subsystem operates in the actual environment. This includes the physical setup of fingerprint and face recognition devices, the push button for opening the door, the solenoid lock, the people counting, and the alert system.

3.5.1.1 Fingerprint Recognition Subsystem

The fingerprint recognition subsystem enables biometric authentication by identifying users based on their unique fingerprint patterns. It is physically implemented using two fingerprint sensors connected to ESP8266 microcontrollers, supporting both enrollment and matching operations. This subsystem plays a key role in the two-phase identity verification process, used at both the entry and exit points to ensure secure access. Here, we present one example of these two phases.

1. **Hardware Overview:**
   - **ESP8266 Microcontroller:**
     It processes fingerprint data locally for low-latency responses and real-time decisions. This decentralized architecture improves performance by handling biometric data near its source.
   - **Fingerprint Sensors:**
     As we explained before, in our system, we have two phases user confirmation mechanism. It means we use two fingerprint sensors as shown in the figure 3.14. The first sensor performs initial identification at entry; the second ensures the entering person matches the authenticated individual. The fingerprint process includes enrollment and matching
     a **Enrollment Process:**
       – User places finger twice on sensor.
       – System processes images to create a fingerprint template.
       – Template stored for comparison, linked to user ID.
     b **Matching Process:**
       – User places finger again on sensor.
       – New fingerprint template generated.
       – System compares template with stored records (1:1 or 1:N matching).

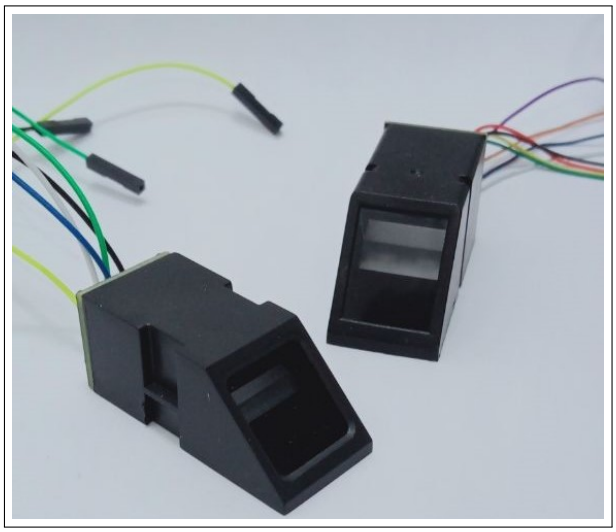


FIGURE 3.14: Fingerprint Sensors

c **Decision Process:**
   - **Matched:** The system confirms identity.
   - **Not matched:** No matching fingerprint is found.

Fingerprint recognition offers secure authentication with the following characteristics [2]:

Advantages:

* High authentication accuracy.
* Fast processing speed and response time.

Disadvantages:

* Relatively expensive compared to traditional authentication methods.
* Potential issues with damaged or unreadable fingerprints.

2. **Fingerprint Authentication subsystem Wiring:**
A fingerprint sensor connects to an ESP8266 board for biometric authentication. Communication via UART (serial) uses digital pins D5 (RX) and D6 (TX). This allows ESP8266 to enroll, detect, and verify fingerprints. Figure 3.15 shows the connection of the fingerprint to the ESP8266.

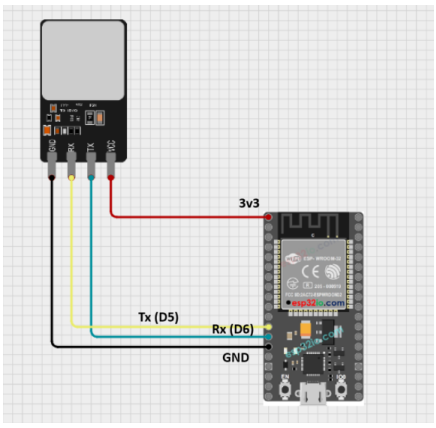


FIGURE 3.15: Wiring of Fingerprint Sensor to the ESP8266.

For better clarity, the table below 3.2 is clearly the fingerprint sensor wiring connections:

Fingerprint Pin	Function	ESP8266 Pin
VCC (Fingerprint)	Power supply	3V3
GND (Fingerprint)	Ground	GND
TX (Fingerprint)	Sensor Transmit (Data Out)	D5 (GPIO14) - ESP Receive (RX)
RX (Fingerprint)	Sensor Receive (Data In)	D6 (GPIO12) - ESP Transmit (TX)

TABLE 3.2: Wiring of Fingerprint Sensor to ESP8266.

3. **Fingerprint Subsystem Physical Assembly:**
In the figure 3.16, we assembled the implemented fingerprint recognition subsystem based on the wiring pre-defined in the table 3.2.

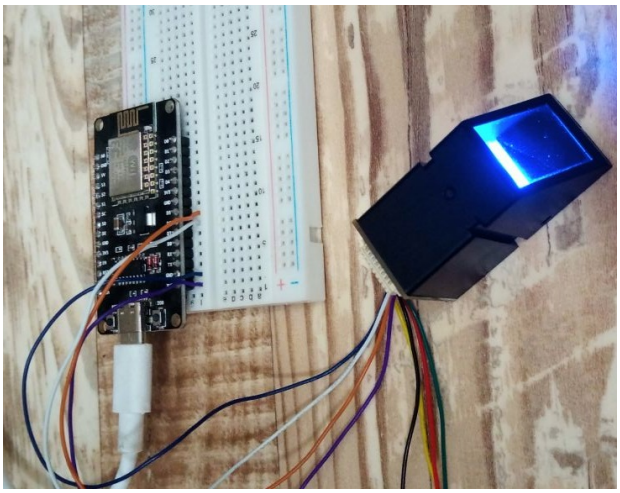


FIGURE 3.16: Physical Assembly of Fingerprint Access Control Subsystem.

3.5.1.2 Facial Recognition Subsystem

The facial recognition subsystem enables biometric authentication by identifying users based on unique facial features. It is physically implemented using a camera module connected to a Raspberry Pi, which runs face detection and recognition algorithms using libraries such as OpenCV and DeepFace. This subsystem capturing facial data for registration and later matching live inputs against stored profiles. Integrated into the two-phase identity verification process, it ensures secure access control.

1. **Hardware Overview:**

   - **Raspberry Pi (Model 4GB or equivalent):**
   Serves as a central edge computing unit for facial recognition and surveillance camera management. It runs lightweight machine learning models for real-time facial authentication and streams camera feeds for continuous monitoring. Integrates with the ESP8266 microcontrollers to cross-validate identity using multiple biometric modalities.

- **Web Cameras (x2):**
  Positioned strategically to capture facial data from different angles to improve recognition reliability and reduce spoofing risks. These USB webcams are connected to the Raspberry Pi.

2. **Facial Recognition Subsystem Wiring :**
   Each Raspberry Pi node is paired with a dedicated USB webcam for facial recognition. These units process video streams locally and publish authentication status to the central surveillance system using the MQTT protocol. This design supports distributed processing and reduces latency. Figure 3.17 shows the connection of the Webcam to the Raspberry pi

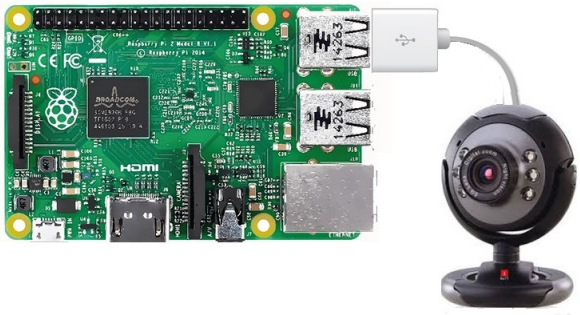


FIGURE 3.17: Facial Recognition Subsystem Using Raspberry Pi and USB Webcam.

- **Raspberry Pi Facial Recognition Node:**
  Table 3.3 shows the major components and their functions within a typical Raspberry Pi-based facial recognition node.

Component	Function	Connection Type
USB Webcam	Captures video for facial input	USB to Raspberry Pi
Raspberry Pi	Processes facial recognition	Wi-Fi / Ethernet
MQTT Client	Publishes recognition status	MQTT over network
Power Supply	Powers Raspberry Pi and webcam	5V 2.5A USB-C or Micro-USB

TABLE 3.3: Raspberry Pi Facial Recognition Node Overview.

3. **Facial Recognition Subsystem Physical Assembly:**
   Figure 3.18 shows the physical setup of the facial recognition subsystem. Each Raspberry Pi unit is connected to a dedicated webcam via USB and powered independently. The Raspberry Pi runs facial recognition software and communicates with the central system over MQTT. This setup operates as a standalone IoT node, publishing recognition results securely.

FIGURE 3.18: Physical Assembly of Facial Recognition Subsystem with Raspberry Pi and Webcam.

3.5.1.3 Open Door Button Subsystem:

In our global system use a push button in the entry phase to trigger door unlocking after successful fingerprint and face recognitions.

1. **Hardware overview:**

   The subsystem consists of the following components:

   - **Push Button:**
     We use a push button to open the door after both fingerprint and face recognitions.
   - **ESP8266 Microcontroller:**
     A second ESP8266 is responsible for detecting push-button presses.

2. **Open Door Button Subsystem Wiring :**
   The wiring of the open door button subsystem is cleared in the table 3.4 below.

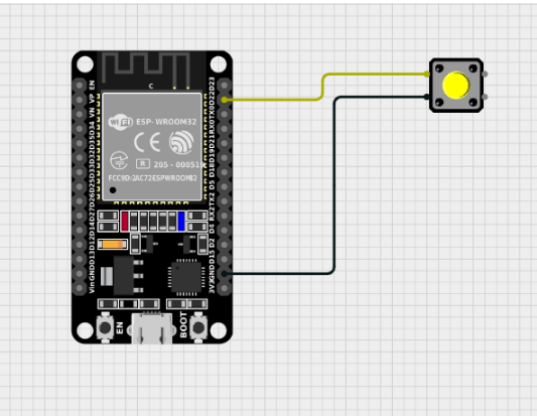


FIGURE 3.19: Wiring of Push Button to the ESP8266.

LED Pin	Function	ESP8266 Pin
One leg of Button	Push button signal input	D4
Other leg of Button	Connected to ground (GND)	GND

TABLE 3.4: Wiring of Push Button to ESP8266.

3. **Open Door Button Subsystem Physical Assembly:**
The real-world implementation of the push button subsystem is shown in Figure 3.20. The push button is mounted on a breadboard and connected to the ESP8266 using jumper wires.

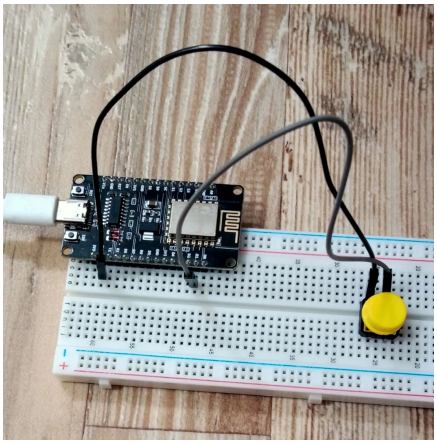


FIGURE 3.20: Physical Assembly of Push Button to the ESP8266.

3.5.1.4 Door Controller Subsystem

The door controller subsystem is responsible for physically unlocking the door after the system authorizes access. It uses a relay module to control a solenoid door lock, interfaced with the ESP8266 microcontroller already used in the push-button subsystem.

1. **Hardware Overview:** It includes the following components:

   - **Relay Module:**
   The relay module (Figure 3.21) is an electronic switch controlling the solenoid door lock based on microcontroller signals.

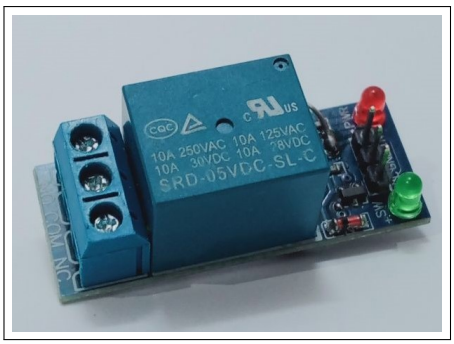


FIGURE 3.21: Relay Module

- **Solenoid Door Lock:**
  (Figure 3.22)It serves as the physical locking mechanism in our smart physical access control system.

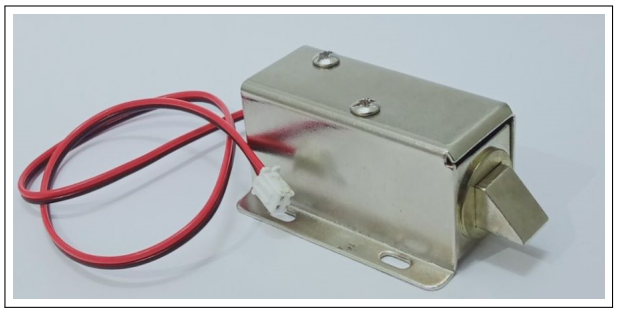


FIGURE 3.22: Solenoid Door Lock

2. **Door Controller Subsystem Wiring :**
   A relay module switches the high current for a solenoid door lock, controlled by the second ESP8266 which we use it in the open door button subsystem. The relay interfaces ESP8266 and the solenoid's external power. Figure 3.23 shows the solenoid and relay module with the ESP8266 wiring.

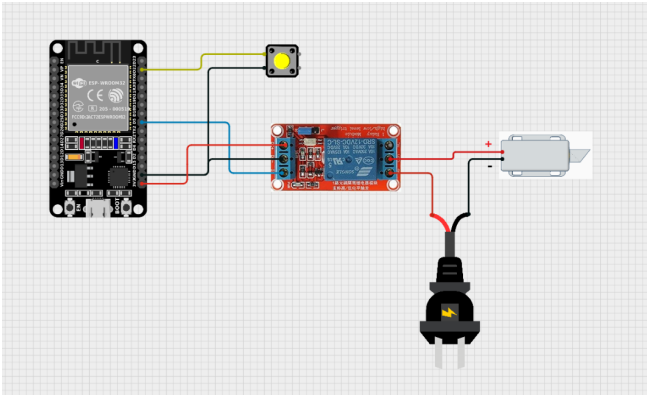


FIGURE 3.23: Wiring Of Solenoid Door Lock with Relay Module to ESP8266.

- **Wiring Relay Module to ESP8266:** Table 3.5 shows the connections between the relay module and the ESP8266.

Relay Pin	Connect To
IN	D1 (GPIO5) on ESP8266
VCC	3.3V on ESP8266
GND	GND on ESP8266

TABLE 3.5: Wiring Of Relay Module to ESP8266.

- **Wiring Relay (Switch Side) to Solenoid Lock and Power Supply:** Table 3.6 shows connections.

Relay Output	Connect To
COM	Solenoid Lock (+)
NO	12V power supply (+)
Solenoid Lock (-)	12V power supply GND (–)

TABLE 3.6: Wiring of Relay Output to Solenoid Lock and Power Supply.

3. **Door Lock Subsystem Physical Assembly:**
The complete door lock subsystem integrates the ESP8266, relay module, solenoid lock, and power source. The physical setup is shown in the figure 3.24.

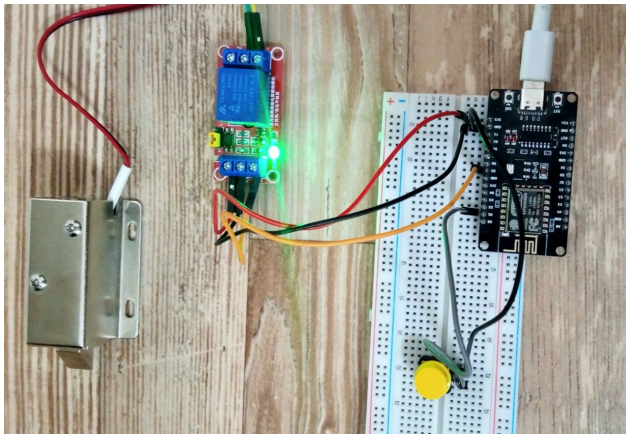


FIGURE 3.24: Physical Assembly Door Lock Subsystem.

3.5.1.5 People Counting Subsystem

The surveillance camera subsystem enables automated people counting by detecting and tracking individuals within a monitored area.

1. **Hardware Overview:**

   - **Surveillance Camera:** Provides continuous video surveillance of the secured area. Streams video to the external storage.

3.5.1.6 Alert Subsystem

In this alert subsystem, we choose an LED as an alert. The alert subsystem is designed to notify users or the system of specific events or unauthorized actions. In our implementation, we use an LED as a visual alert indicator connected to the ESP8266 microcontroller.

1. **Hardware Overview:**
   LED:
   We use a standard LED as the alert indicator connected to the ESP8266 microcontroller.

2. **Alert Subsystem Wiring:**
   The LED connects to digital pin D2 of the ESP8266 board. The figure 3.25 shows the LED wiring with the microcontroller, while the table 3.7 details the wiring configuration, where the black wire is the GND.

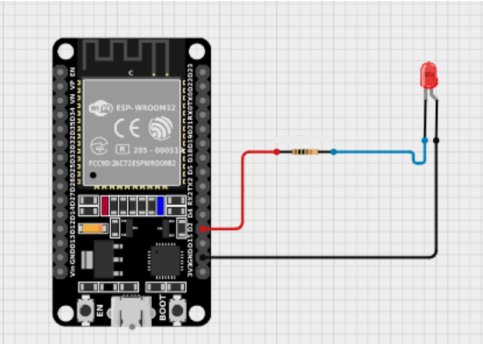


FIGURE 3.25: Wiring of LED alert with ESP8266..

Component Pin	ESP8266 Pin
Long leg (Anode) of LED	D2
Short leg (Cathode) of LED	GND

TABLE 3.7: Wiring of LED as Alert to ESP8266.

3. **Alert Subsystem Physical Assembly:**
   The physical setup of the alert subsystem is shown in Figure 3.26. The LED is mounted on a breadboard and connected directly to the ESP8266 using jumper wires.

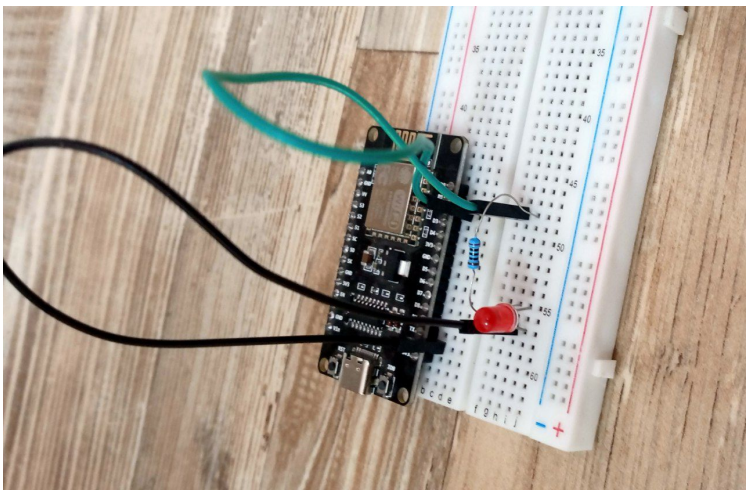


FIGURE 3.26: Physical assembly of the LED alert subsystem with ESP8266.

3.5.2 Software Implementation

In this subsection we present the System Manager and the Door controller System and communication implementation

3.5.2.1 System Manager

In this section, we present the use cases of the System Manager, along with their descriptions, frontend production and test code, screenshots of the web application's UI, and backend production and test code.

Use Cases

Name	**Description**
Add Role	Defines new system roles.
Delete Role	Removes a role after reassignment or revocation.
Edit Role	Updates role properties or permissions.
Add User	Registers new users.
Delete User	Permanently removes a user account.
Edit User	Modifies user information.
Search User	Locates users by attributes with filtering.
Assign User Role	Associates users with roles.
Get Access Log	Retrieves access logs (door access, fingerprint scans, face detection).
Get Roles	Retrieves all registered roles.
Get Users	Retrieves all registered users.
Sign In	Authenticates users.
Sign Up	Allows new user registration.
Logout	Terminates user session.
Stream View	Displays live video from a user's camera.
Customer Monitoring	Tracks customer entry and exit.

Table 3.8: System Manager Use Cases

3.5.2.2 Frontend Implementation

The frontend development followed modern web development practices with a focus on maintainability, scalability, and user experience. The implementation adopted component-based architecture combined with rigorous testing methodologies to ensure code quality throughout the development lifecycle.

UI Component Tests

```
1  it("should render logo, inputs, and button", () => {
2      const onSubmit = jest.fn();
3      render(SignInFormater.format({ onSubmit }));
4
5      expect(screen.getByAltText("Logo")).toBeInTheDocument();
6      expect(screen.getByText("Access.com")).toBeInTheDocument();
7      expect(
8        screen.getByPlaceholderText(/Enter your Username/i),
9      ).toBeInTheDocument();
10     expect(
11       screen.getByPlaceholderText(/Enter your password/i),
12     ).toBeInTheDocument();
13     expect(screen.getByRole("button", { name: /Log in/i
       ↪  })).toBeInTheDocument();
14   });
```

Listing 15: Frontend UI Test: Rendering of Sign In Page Elements

User Interface Pages

Figures 3.27-?? show the System Manager's UI for Statistics, Assign Roles, Customer Management, and Activity History pages.

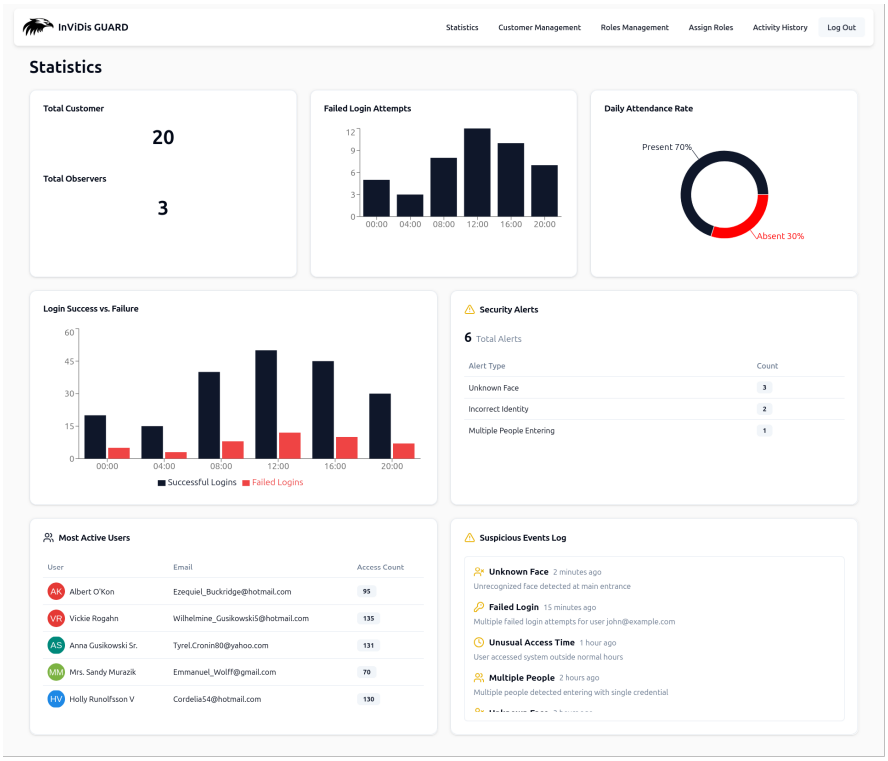


FIGURE 3.27: User Interface: Statistics Page

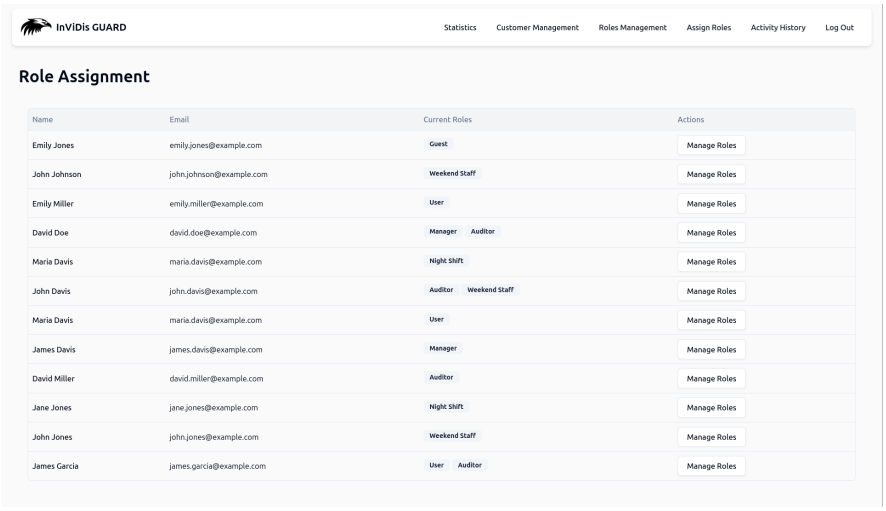


FIGURE 3.28: User Interface: Assign Roles Page

3.5.2.3 Backend Implementation

The System Manager backend, a robust API for surveillance data processing and authorization, prioritized modularity, testability, and separation of concerns for maintainability and scalability.

Add Role Use Case

```
class AddRoleUseCase(AddRoleUseCaseInterface):
    def __init__(self, role_saver_service: RoleSaverServiceInterface):
        self.role_saver_service: RoleSaverServiceInterface =
        ↪ role_saver_service

    def execute(self, add_role_dto: AddRoleDTO) ->
    ↪ AddRoleUseCaseResponseDTO:
        role: Role = Role(name=add_role_dto.role_name)
        self.role_saver_service.save(role)
        return AddRoleUseCaseResponseDTO(status="Success")

```

Listing 16: Add Role Use Case Code

```
class RoleDuplicationCheckerAspect(AspectInterface):
    def __init__(
        self, role_existence_checker_service:
        ↪ RoleExistenceCheckerServiceInterface
    ):
        self.role_existence_checker_service:
        ↪ RoleExistenceCheckerServiceInterface = (
            role_existence_checker_service
        )

    def apply(self, _func: Callable[..., Any]) -> Any:

        @functools.wraps(_func)
        def wrapper(add_role_dto: AddRoleDTO, *args, **kwargs):
            def __is_role_duplicated(_role: Role) -> bool:
                return
                ↪ self.role_existence_checker_service.role_exists(_role)

            role: Role = Role(name=add_role_dto.role_name)
            if __is_role_duplicated(role):
                return AddRoleUseCaseResponseDTO(status="Failed",
                ↪ duplicated_role=True)
            else:
                return _func(add_role_dto, *args, **kwargs)

        return wrapper

```

Listing 17: Role Duplication Checker Aspect Code

```python
unregistered_roles_parametrize_data: list[AddRoleDTO] = [
    AddRoleDTO(role_name="admin"),
    AddRoleDTO(role_name="user"),
    AddRoleDTO(role_name="moderator"),
    AddRoleDTO(role_name="editor"),
    AddRoleDTO(role_name="viewer"),
]

@pytest.mark.parametrize(
    "unregistered_roles",
    unregistered_roles_parametrize_data,
    ids=generate_case_ids(unregistered_roles_parametrize_data),
)
@pytest.mark.asyncio
def test_add_unregistered_role_return_success_response(
    client: TestClient, unregistered_roles: AddRoleDTO
):
    # Arrange (Given)
    expected_response = {"status": "Success", "duplicated_role": False}
    request_dto: dict = {"role_name": unregistered_roles.role_name}

    # Act (When)
    actual_response = client.post("/add_role", json=request_dto).json()

    # Assert (Then)
    assert actual_response == expected_response

```

Listing 18: Add Role Use Case Acceptance Test

```
add_role_parametrize_data: list[AddRoleDTO] = [
    AddRoleDTO(role_name="viewer"),
    AddRoleDTO(role_name="contributor"),
    AddRoleDTO(role_name="manager"),
    AddRoleDTO(role_name="analyst"),
    AddRoleDTO(role_name="developer"),
    AddRoleDTO(role_name="support"),
]

@pytest.mark.parametrize(
    "add_role_dto",
    add_role_parametrize_data,
    ids=generate_case_ids(add_role_parametrize_data),
)
def test_add_non_existing_role_calls_use_case_with_expected_dto(
    add_role_dto: AddRoleDTO,
):
    def __assert_use_case_was_called_with_expected_dto(
        _spy_use_case: Mock, _add_role_dto: AddRoleDTO
    ) -> None:
        _spy_use_case.execute.assert_called_once_with(_add_role_dto)

    # Arrange (Given)
    spy_use_case: Mock = Mock(spec=AddRoleUseCaseInterface)
    stub_role_existence_service: Mock = create_stub_role_existence_service(
        role_exists=False
    )

    decorated_use_case: DecoratedAddRoleUseCase =
    ↪ build_decorated_add_role_use_case(
        spy_use_case, stub_role_existence_service
    )

    # Act (When)
    decorated_use_case.execute(add_role_dto)

    # Assert (Then)
    __assert_use_case_was_called_with_expected_dto(spy_use_case,
    ↪ add_role_dto)
```

Listing 19: Role Duplication Checker Aspect Test

3.5.2.4 Door Controller System

After we present the system manager, we now move in this section to the implementation of the door controller system.

Fingerprint Recognition Subsystem Implementation

1. **Fingerprint Recognition Subsystem Overview:** The fingerprint subsystem enables secure biometric identification using an fingerprint sensor connected to an ESP8266 microcontroller. It supports both enrollment and matching, while communicating outcomes via MQTT for remote monitoring and integration.

2. **Fingerprint Recognition Process:**

   (a) *Enrollment Process:*
   Enrollment starts when the system receives a message on the topic fingerprint/command with the format:

   ```
   enroll:<id>
   ```

   This is handled in the MQTT callback function:

```
void callback(char* topic, byte* payload, unsigned int length) {
String message;
for (unsigned int i = 0; i < length; i++) {
  message += (char)payload[i];
}

Serial.print("Message arrived [");
Serial.print(topic);
Serial.print("]: ");
Serial.println(message); monitor

if (message.startsWith("enroll:")) {
  int id = message.substring(7).toInt();
  fingerprintSensor.enrollFingerprint(id, feedbackHandler);
}
}
```

Listing 20: Enroll MQTT Calback Function

   The **enrollFingerprint(id, feedbackHandler)** function:

   i. Asks the user to place their finger twice.
   ii. Captures and processes the images.
   iii. Stores the fingerprint template under the given id.

   here an exemple:

```
Message arrived [fingerprint/command]: enroll:4
Place finger on sensor...
Remove finger...
Place same finger again...
Fingerprint stored successfully
```

   (b) *Matching and Decision Proccesses:*
   The loop continuously scans for fingerprints:

```
        int id = fingerprintSensor.getFingerprintID(feedbackHandler);
         sendFingerprintID(id, feedbackHandler);
```

Listing 21: Enroll MQTT Calback Function

- getFingerprintID() captures a new fingerprint.
- It compares it to stored templates.
- If matched, an ID is returned.
- sendFingerprintID(id, feedbackHandler) sends the result via MQTT (topic: auth-out/fingerprint:

```
    void sendFingerprintID(int id, void (*feedbackHandler)(const String&)){
     if (id > 0) {
       String msg = String(id);
       feedbackHandler(msg);
     }
   }
```

Listing 22: Get then send the fingerprint ID to MQTT

```
  void feedbackHandler(const String& message) {
  Serial.println(message);
  mqttClient.publish(fingerprint_topic, message.c_str());
}
```

Listing 23: FeedbackHandler Function

Facial Recognition Subsystem Implementation

This details the real-time facial recognition subsystem: model selection, modules (detection, alignment, preprocessing, embedding, recognition), workflow, and performance.

1. **Overview** A real-time facial recognition system for access control balances accuracy, speed, and resource efficiency. After evaluating face detection and embedding models (metrics: accuracy, inference speed, resource use), a modular pipeline was selected: OpenCV's Yunet face detector, preprocessing (alignment, normalization), and DeepFace's Facenet model for feature extraction. Recognition uses cosine similarity on face embeddings and a KNN classifier, robust to pose, lighting, and occlusion variations, with tunable parameters (neighbor count, similarity threshold).

   - **Motivation for Model Selection**
     Low-latency and efficiency are critical. Models balancing performance and resource use were favored over high-accuracy models with poor performance or high memory demands. The selected pipeline ensures high accuracy and efficiency for mid-range hardware.
   - **Tested Models and Evaluation**
     Table 3.9 compares tested face detection and embedding models.

Component	Model	Accuracy	Resource Consumption	Notes
Face Detection	Haar Cascade	Low	Low	Fast but inaccurate under varied lighting and poses.
	Dlib CNN	Medium	High	Accurate, but slow inference, especially on CPU-only systems.
	OpenCV Yunet	**High**	**Medium**	Balanced speed and accuracy; includes landmark detection simplifying alignment. Selected.
Embedding Model	VGG-Face (DeepFace)	Medium	High	Good embedding quality but high memory and compute requirements.
	OpenFace	Low	Medium	Lightweight but low accuracy on our test set.
	Facenet (DeepFace)	**High**	**Medium**	Best balance of accuracy and speed in our tests. Selected.
	ArcFace (InsightFace)	**High**	**High**	State-of-the-art accuracy but slower inference times.
	SFace (InsightFace)	Medium	Medium	More efficient than ArcFace but slightly lower accuracy.

TABLE 3.9: Comparison of face detection and embedding models evaluated.

2. **System Modules**

- **Face Detection and Alignment**
  OpenCV's Yunet model (`face_detection_yunet_2023mar.onnx`) is used for fast and accurate detection, including landmark identification. The alignment step uses a custom `FaceAligner` class:
  - Detect faces and facial landmarks.
  - Extract eye coordinates.
  - Align face via rotation and scaling.
  - Crop the aligned face for further processing.

  Rationale: Integrated landmark detection simplifies preprocessing and improves recognition reliability.
- **Image Preprocessing**
  Images are standardized before embedding using resizing and normalization:
  - **Resizing:** All images resized to 160x160 to match Facenet input.

– **Normalization:** Pixel values normalized via mean subtraction and scaling.

Code Example:

```
from PIL import Image
import numpy as np
import os

class ImageResizer:
    def __init__(self, size: tuple[int, int] = (160, 160)):
        self.size = size

    def resize_image(self, image_path: str) -> Image.Image:
        if not image_path or not os.path.exists(image_path):
            raise ValueError(f"Invalid or missing image path: {image_path}")
        image = Image.open(image_path)
        resized_image = image.resize(self.size, Image.Resampling.LANCZOS)
        return resized_image

class ImageNormalizer:
    def __init__(self, mean: float = 127.5, scale: float = 128.0):
        self.mean = mean
        self.scale = scale

    def normalize(self, image: Image.Image) -> np.ndarray:
        img_array = np.asarray(image).astype(np.float32)
        normalized = (img_array - self.mean) / self.scale
        return normalized
```

Listing 24: ImageResizer and ImageNormalizer classes

- **Embedding Extraction**
  Embeddings are generated with DeepFace using the Facenet model:

  Note: Detection is disabled since faces are already aligned.
- **Face Recognition**
  Recognition is handled by a KNN classifier:
  – Load known embeddings and identities.
  – Extract embedding from input face.
  – Compute cosine similarity.
  – Apply $k = 7$ KNN majority voting.
  – Output identity if similarity exceeds threshold (e.g., 0.6).

  Details: KNN improves robustness to variability and noise. The similarity threshold is configurable to balance precision and recall.

3. **Summary of Workflow**

4. **Output and Performance**

- **Example Output:**
- **Performance Metrics:**
  Achieves over 80% accuracy on test set; inference latency under 40 ms/frame

```python
from typing import List, Union
from deepface import DeepFace

class FeatureExtractor:
    def __init__(self, model_name: str = "Facenet"):
        self.model_name = model_name

    def extract_embedding(self, image_path: str) -> Union[List[float], None]:
        try:
            results = DeepFace.represent(
                img_path=image_path,
                model_name=self.model_name,
                enforce_detection=False
            )
            if isinstance(results, list) and len(results) > 0:
                return results[0]["embedding"]
            else:
                return None
        except Exception as e:
            print(f"[Error] Failed to extract embedding: {e}")
            return None
```

Listing 25: extract Embeddings from face function

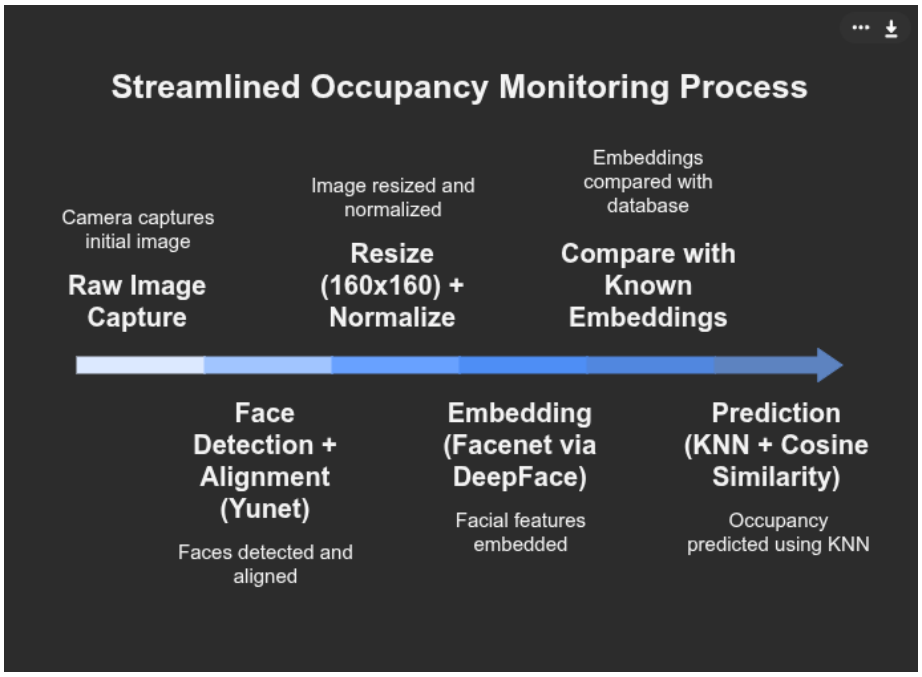


Figure 3.29: Pipeline of the facial recognition system

```
Detected: 550e8400-e29b-41d4-a716-446655440000 (average similarity: 0.8449)
Detected: 550e8400-e29b-41d4-a716-446655440000 (average similarity: 0.8618)
Detected: 550e8400-e29b-41d4-a716-446655440000 (average similarity: 0.8553)
```

Listing 26: example of output

(mid-range CPUs). Moderate resource use supports embedded/edge deployment.

Open Door Button Subsystem Implementation

Our system includes a push-button. When the button is pressed, the system publishes the MQTT message with payload:

```
Button pressed
```

via MQTT topic: "auth-out/click-btn" to notify subscribers.

- *Button Setup*:

```
PressButton button(pushBtn_pin);
```

- *Button Press Handling* In the main loop, the system continuously checksthe push-button using button.isPressed(). When pressed, it publishes the message "Button pressed" to the MQTT topic:"auth-out/click-btn", notifying subscribed devices in real time.

```
if (button.isPressed()) {
    mqttClient.publish(button_topic, "Button pressed");
}

```

Listing 27: PressButton class

Door Lock Subsystem Implementation
The system controls a door lock using messages payload="UNLOCK" it receives through MQTT on the topic:"auth/open-door".

1. *Startup and Setup:* When the system starts, After connects to Wi-Fi and MQTT broker, It subscribes to the MQTT topic:"auth/open-door".
2. *Unlock commands from subscribed auth/open-door topic:* Whenever a new message arrives on the topic auth/open-door:

```
mqttClient.subscribe(door_sub_topic );à
```

If the message is "UNLOCK", the system unlocks the door Waits for 10 seconds then locks the door again.

```
if (message == "UNLOCK") {
    door.unlock();
    delay(10000);
    door.lock();
}

```

Listing 28: Unlocking the door upon receiving the "UNLOCK" command via MQTT

People Counting Subsystem Implementation:
This details the real-time people counting system: overview, motivation, architecture (camera, MQTT, detection), workflow, performance.

```
results = model(frame, classes=[0])
count = 0
for result in results:
    for box in result.boxes:
        cls_id = int(box.cls[0])
        label = model.names[cls_id]
        if label == 'person':
            count += 1
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

Listing 29: detect person class

1. **Overview** A real-time people counting system improves access control and environmental monitoring. It uses YOLOv8 object detection and MQTT messaging for responsive communication. Designed for edge devices, it supports smart surveillance and automated building systems. Camera captures images; upon MQTT command, YOLOv8n processes frame, detects people, returns count via MQTT.

   - **Motivation for Design Choices**
     YOLOv8n (Nano) chosen for compactness and real-time efficiency on resource-constrained hardware. MQTT chosen for low overhead and IoT publish-subscribe suitability.

2. **System Architecture and Modules**

   - **Camera Initialization**
     Camera initialized via OpenCV's `VideoCapture`, with validation. Exits gracefully if camera unavailable.
   - **MQTT Communication Setup**
     The MQTT client is configured using the `paho.mqtt.client` library with the following parameters:
     - **Broker:** `mosquitto`
     - **Port:** 1883
     - **Subscribe Topic:** `people/count/request`
     - **Publish Topic:** `people/count/response`

     Upon receiving a trigger message on the subscribed topic, the system processes a fresh frame and returns the detected count.
   - **Detection Pipeline**
     The detection module utilizes YOLOv8n pretrained on the COCO dataset. Only class ID `0` (person) is detected for performance optimization.
     Code Snippet: Robustness Measures:
     - Skips the first 5 frames to avoid stale data.
     - Validates captured frame dimensions and detection class indices.
     - Implements exception handling to maintain operational stability.
   - **Output and Message Publishing**
     The detected count is sent via MQTT to the `get_people_count` topic:

     ```
     client.publish(topic_pub, str(count))
     ```

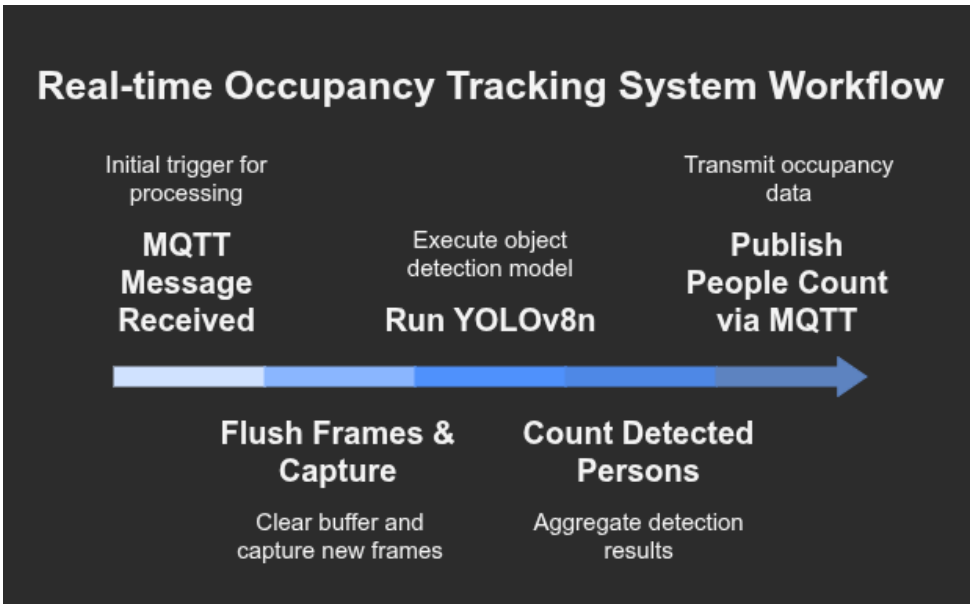


FIGURE 3.30: Pipeline of the people counting system

```
Received MQTT message on people_service_listener: trigger
Detected people: 3
Published to get_people_count: 3
```

LISTING 30: example of output

3. **Summary of Workflow**

4. **Example Output and Performance**

   - **Example Output:**

- **Performance Metrics:**
  Real-time operation; average latency under 100 ms (mid-range CPU). YOLOv8n efficiency and frame filtering ensure low memory/CPU use, suitable for embedded platforms.

Alert Subsystem Implementation:
The system activates an LED alert using the message payload="Active" it receives through MQTT on the topic: "alert/active".

1. *Startup and Setup:* When the system starts, after connecting to Wi-Fi and the MQTT broker, it subscribes to the MQTT topic: "alert/active".
2. *Activating the alert LED from the subscribed alert/active topic:* Whenever a new message arrives on the topic "alert/active":

   ```
   mqttClient.subscribe(alert_sub_topic);
   ```

   If the message is "Active", the system makes the LED blink (on/off repeatedly) for 10 seconds.

```
1    if (message == "Active") {
2        led.activate();
3    }
```

Listing 31: Activating the alert LED upon receiving the "Active" command via MQTT

3.5.2.5 Core Logic

This part details the system's core logic,core design, Clark-Wilson Integrity Model adoption, use cases, and implementation.
Core Design

The core design divides the system according to the architecture outlined in Figure 2.1.

1. **Execution Environment:** Manages real-time physical interactions: facial/fingerprint authentication (outdoor/indoor), door lock operation, and people counting.
2. **Decision Environment:** Handles higher-level logic: authenticating users against a central database, determining access authorization based on policies, and verifying people count integrity from the Execution Environment.

Clark-Wilson Security Model Adoption

When adopting the Clark-Wilson Integrity Model for implementation within a Door Controller System, we derive the following mappings based on the core components of the model:

- **Constrained Data Item**: The physical door is modeled as a Constrained Data Item, representing the protected resource whose integrity must be preserved. Access and modification operations on the door state must adhere to defined integrity constraints.

- **Transformation Procedure**: The act of opening the door is represented as a Transformation Procedure. This operation enforces controlled and validated changes to the CDI (i.e., the door), ensuring that any state transition occurs only through authorized and well-defined procedures.
- **User (Subject)**: An authenticated and authorized individual who is permitted to initiate the transformation procedure (i.e., open the door). The user must be validated by the system according to predefined access control rules.
- **Integrity Verification Procedures**: These include mechanisms such as dual fingerprint authentication verification, dual facial recognition checks, people count validation, and time interval analysis between authentication attempts. These procedures ensure that the system's critical data and access control decisions remain valid and untampered.

This mapping illustrates how the Clark-Wilson Model can be effectively applied to real-world systems to enforce data integrity and ensure that all modifications to critical resources occur through controlled and auditable processes.

Core Use Cases

Name	**Description**
Outdoor Face Authentication	Authenticates user's face outside before entry.
Outdoor Fingerprint Authentication	Authenticates user's fingerprint outside before entry.
Open Door	Activates door lock after successful multi-factor authentication and user presses open door button.
Indoor Face Authentication	Authenticates user's face inside after entry.
Indoor Fingerprint Authentication	Authenticates user's fingerprint inside after entry.

Table 3.10: Core Logic Use Cases

Door Controller System Integrity Verification Procedures

In this section, we explain the Integrity Verification Procedures (IVPs) used in our system:

1. **None ID IVP:** Checks whether any provided `ID` is `None`. If so, it raises a `InvalidStateIVPException`.
2. **Same User Authentication IVP:** Ensures that all authenticated IDs match—confirming that the same user authenticated at both the exterior and interior. If they don't match, it raises a `InvalidStateIVPException`.
3. **Authentication Time Interval IVP:** Verifies that the time between **opening the door** and completing **interior authentication** does not exceed a predefined threshold.
4. **People Count IVP:** Verifies that
   `people count after user enter = people count before enter + 1`

Some IVP Code Implementation:

```
class IVPInterface(metaclass=ABCMeta):
    @abstractmethod
    def verify(self):
        """abstract method"""

```

Listing 32: IVP Interface

```python
class SameUserAuthenticateIVP(IVPInterface):
    def __init__(self, authentication_repository:
    ↪  AuthenticationRepositoryInterface):
        self.authentication_repository: AuthenticationRepositoryInterface =
        ↪  (
            authentication_repository
        )

    def verify(self):
        def __ids_are_different(_ids):
            return len(set(_ids)) != 1

        exterior_fingerprint_id =
        ↪  self.authentication_repository.get("out-fingerprint")[
            "id"
        ]
        exterior_face_id =
        ↪  self.authentication_repository.get("out-face")["id"]
        interior_fingerprint_id =
        ↪  self.authentication_repository.get("in-fingerprint")[
            "id"
        ]
        interior_face_id =
        ↪  self.authentication_repository.get("in-face")["id"]
        ids = [
            exterior_fingerprint_id,
            exterior_face_id,
            interior_fingerprint_id,
            interior_face_id,
        ]

        if __ids_are_different(ids):
            raise InvalidStateIVPException("Invalid state")

```

Listing 33: Same User Authentication IVP

```python
class AuthenticationTimeIntervalIVP(IVPInterface):
    def __init__(
        self,
        authentication_repository: AuthenticationRepositoryInterface,
        threshold: int,
    ):
        self.authentication_repository: AuthenticationRepositoryInterface =
        ↪  (
            authentication_repository
        )
        self.threshold: int = threshold

    def verify(self):
        open_door_time =
        ↪  self.authentication_repository.get("open_door_time")["time"]
        inner_fingerprint_authentication_time =
        ↪  self.authentication_repository.get(
            "in-fingerprint"
        )["time"]

        diff_seconds = (inner_fingerprint_authentication_time -
        ↪  open_door_time).seconds

        if diff_seconds > self.threshold:
            raise InvalidStateIVPException("Invalid state")

```

Listing 34: Authentication Time Interval

3.5.3 Communication Architecture

This section explains how different components in the system exchange information using the MQTT protocol. We use it to connect the microcontrollers, sensors, actuators, and the core logic.

3.5.3.1 MQTT Broker and Topics

1. **MQTT Broker:** we use mosquitto
2. **MQTT Topics:**
   Each component uses specific MQTT topics to either publish messages (send) or subscribe to them (receive). we orgnize them in the table 3.11:

Component	MQTT Role	MQTT Topic
Fingerprint Recognition Out	Publish	`auth-out/fingerprint`
Face Recognition Out	Publish	`auth-out/face`
Press Button	Publish	`auth-out/click-btn`
Core Logic	Subscribe	`auth-out/fingerprint` and `auth-out/face`
Core Logic	Subscribe	`auth-out/click-btn`
Fingerprint Recognition In	Publish	`auth-in/fingerprint`
Face Recognition In	Publish	`auth-in/face`
Core Logic	Subscribe	`auth-in/fingerprint` and `auth-in/face`
Core Logic	Publish	`people/count/request`
People Counter	Subscribe	`people/count/request`
People Counter	Publish	`people/count/response`
Core Logic	Subscribe	`people/count/response`
Core Logic	Publish	`auth/open-door`
Door Controller (Lock)	Subscribe	`auth/open-door`
Alert Active	Publish	`alert/active`
Alert Active	Subscribe	`alert/active`

TABLE 3.11: MQTT Topics Used in the System

3. **Connect and Callback MQTT functions:**
   - **MQTT Connection Function:**
     It establishes connection to the MQTT broker, subscribes to relevant topics, and initializes communication parameters for fingerprint system control

```
void connectMQTT() {
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str(), mqtt_user, mqtt_password)) {
      Serial.println("Connected to MQTT broker");
      client.subscribe(topic);

    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retrying in 5 seconds");
      delay(5000);
    }
  }
}
```

Listing 35: MQTT Connection Setup Function

- **MQTT Callback Function:**
  It receives messages, reads commands, and runs actions based on the message.

```
void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("]: ");
  Serial.println(message);

  if (message.startsWith(payload)) {
    int id = message.substring(7).toInt();
    fingerprintSensor.enrollFingerprint(id, feedbackHandler);
  }
}
```

Listing 36: MQTT Callback Function

3.5.3.2 System Communication Diagram

The overall communication architecture of the system using MQTT illustrates in the figure 3.31:

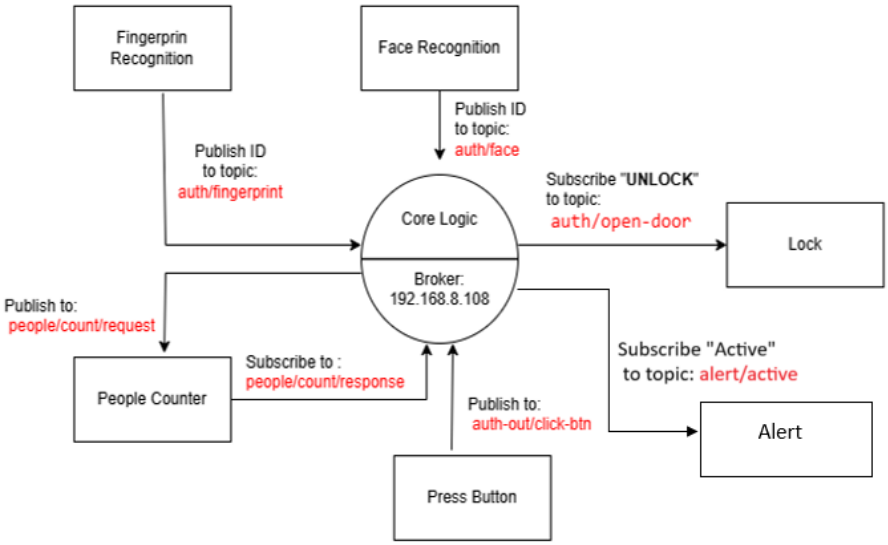


FIGURE 3.31: MQTT Communication Diagram.

3.6 Conclusion

This conclusion summarizes the design and development of the proposed IoT-based intelligent surveillance system, which integrates biometric technologies, IoT devices, computer vision modules, and the Clark-Wilson security model. The chapter detailed each component's role and its interaction with other components through secure protocols to ensure controlled access and real-time monitoring, reiterating the functions of its key components: the System Manager and the Door Controller System. The chapter outlined the design and implementation details of this smart access control system, focusing on the following key components:

- **System Manager:** The chapter detailed this central server component, which is responsible for identity verification, enforcing access control policies based on the Clark-Wilson model, facilitating real-time monitoring, and managing data logging.
- **Door Controller System:** The chapter explained the hardware setup for this system, which encompasses the integration of fingerprint and facial recognition systems, smart lock control mechanisms, and the YOLO-based people counting module.

During the development of our smart access control system, we encountered several challenges and limitations:

- **Limitations:**
  The several limitations were encountered:

  - Budget constraints prevented the use of advanced or industrial-grade components.
  - Run high-performance models on cheap devices.
  - Lack of electrical knowledge caused hardware damage and material loss.
  - Took time to find a solution to use our relay with ESP's 3.3V pins.

- **Challenges:**
  In addition to these limitations, the development process posed several challenges:

  - Finding the right ai model.
  - Tried to build a circuit as a solution for the relay issue, but it didn't work.
  - Finding the right relay type compatible with ESP 3.3V pins.
  - Link different systems with different characteristics.
  - Convert and modify the old theory model to a real implementation.
  - Develop a complex system in a short period of time

General Conclusion

In conclusion of this study of a smart access control system, we have addressed the main challenges outlined at the beginning by designing a multi-layered security system that combines dual biometric authentication, real-time monitoring, and the Clark-Wilson security model. This work enabled us to explore the practical integration of IoT technologies in physical security contexts and gain hands-on experience with both hardware and software components under constrained conditions. Moreover, team spirit and collaboration were among the key factors that contributed to the success of this project.

As future works:

1. Develop the system into a full-scale surveillance platform capable of managing access and auditing across multiple facilities using a centralized database.
2. Strengthen fingerprint and facial recognition accuracy by integrating advanced anti-spoofing techniques.
3. Additional IVPs can be incorporated into the system utilizing the Clark-Wilson Library, depending on the specific context and security requirements.
4. Use blockchain technology to securely log access events and audit trails.

The outcomes of this research provide valuable insights into IoT-based physical access control. Our analysis demonstrates that the Clark-Wilson Integrity Model is applicable to and can be successfully implemented within such systems. We hope these results will help pave the way for broader practical deployment of secure and monitored access control solutions in sensitive environments requiring a high level of security.

System Vulnerability Assessment

The system's multi-layered architecture makes it highly resistant to conventional attacks, though not impervious:

Security Strengths

- **Dual biometric authentication:** Requires simultaneous spoofing of both facial and fingerprint modalities.
- **Clark-Wilson enforcement:** Implemented via IVPs (Identity, Time Interval, Same User, People Count), which trigger automatic rollback upon policy violations.
- **Tamper-proof logs:** Secured using AES-256 encryption with real-time anomaly alerts.

Residual Risks

- **DeepFake attacks:** May bypass facial recognition using advanced synthetic media techniques.
- **Hardware tampering:** Possible manipulation of people-counting camera angles (e.g., employing mannequins).
- **Cost of attack:** Estimated at over $85,000 for coordinated physical and digital breaches.

Ethical Framework

Ethical Framework

The system adheres to human-centric security principles:

Privacy Preservation

- **Local biometric processing:** Biometric data is processed on edge devices; cloud transfer occurs only with explicit user consent.
- **Right-to-erasure compliance:** User data is deleted upon access revocation.

Transparency & Fairness

- **Real-time notifications:** Users receive immediate alerts for access denials (e.g., *"Rejected: Unauthorized tailgating detected"*).
- **Bias-mitigated AI:** Models are trained on diverse datasets, including variations across ethnicities, ages, and genders.

Purpose Limitation

- **Usage constraints:** Explicit prohibition of mass surveillance and discriminatory profiling.

List of Abbreviations

IoT	**I**nternet **o**f **T**hings
AI	**A**rtificial **I**ntelligence
CCTV	**C**losed-**C**ircuit **T**ele**V**ision
IP	**I**nternet **P**rotocol
DVR	**D**igital **V**ideo **R**ecorders
NVR	**N**etwork **V**ideo **R**ecorders
YOLO	**Y**ou **O**nly **L**ook **O**nce
M2M	**M**achine-to-**M**achine
IoMT	**I**nternet **o**f **M**edical **T**hings
SSD	**S**ingle-**S**hot multibox **D**etection
VPU	**V**ision **P**rocessor **U**nit
SS-HSA	**S**urveillance **S**ystem for **H**igh-**S**ecurity **A**reas
GMS	**G**ravity **M**icrowave **S**ensor
GSM	**G**lobal **S**ystem for **M**obile Communications
CV	**C**omputer **V**ision
OCR	**O**ptical **C**haracter **R**ecognition
CNN	**C**onvolutional **N**eural **N**etworks
TP	**T**ransformation **P**rocedure
CDI	**C**onstrained **D**ata **I**tems
IVP	**I**ntegrity **V**erification **P**rocedures
UDI	**U**nconstrained **D**ata **I**tems
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
ID	**id**entifier
KNN	**K**-**N**earest **N**eighbors
USB	**U**niversal **S**erial **B**us
API	**A**pplication **P**rogramming **I**nterface
TDD	**T**est **D**riven **D**evelopment
UI	**U**ser **I**nterface
ACS	**A**ccess **C**ontrol **S**ystems
XP	E**x**treme **P**rogramming
QoS	**Q**uality **o**f **S**ervise

Bibliography

[1] José Joaquín Peralta Abadía, Christian Walther, Ammar Osman, and Kay Smarsly. A systematic survey of internet of things frameworks for smart city applications. *Sustainable Cities and Society*, 83:103949, 2022.

[2] Oussama Afouf and Sofia Kouah. Développement d'un système d'iot (internet of things) dans le cadre de smart university. *Revue de l'Université Oum El Bouaghi*, 2020.

[3] Hina Afreen, Muhammad Kashif, Qaisar Shaheen, Yousef H Alfaifi, and Muhammad Ayaz. Iot-based smart surveillance system for high-security areas. *Applied Sciences*, 13(15):8936, 2023.

[4] Radouan Ait Radouan Ait Mouha et al. Internet of things (iot). *Journal of Data Analysis and Information Processing*, 9(02):77, 2021.

[5] Vian S Al-Doori, Mohammed Abdul Jaleel Maktoof, Abdulqader Faris Abdulqader, and Serhii Lienkov. Securing smart buildings using rfid and fingerprint technologies. In *2024 35th Conference of Open Innovations Association (FRUCT)*, pages 71–81. IEEE, 2024.

[6] Rabab Al-Zaidi, John Woods, Mohammed Al-Khalidi, Khattab M Ali Alheeti, and Klaus McDonald-Maier. Next generation marine data networks in an iot environment. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 50–55. IEEE, 2017.

[7] APS (Radio Algérienne) and Reuters via Al Arabiya. Reports on oran airport stowaway incident and legal actions, 2024. Stowaway found in landing gear of Air Algérie flight; 10 detained. Incident occurred Dec 28, 2023, reports published Jan 8, 2024.

[8] Maria Teresa Baldassarre, Danilo Caivano, Davide Fucci, and Burak Turhan. How effective is test driven development? *Empirical Software Engineering*, 26(4):1–38, 2021.

[9] Kent Beck. *Test-Driven Development: By Example*. Addison-Wesley, 2003.

[10] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[11] Harry Bryant and Zoltan Erdos. Ethical and practical challenges of ai in surveillance: A review. *AI and Ethics*, 1(1):45–60, 2021.

[12] J. L. Burch and R. W. Brown. *Comparative Analysis of Access Control Models*. Springer, New York, 2nd edition, 2018.

[13] M. A. G. Carli and D. J. Saad. Evolution of access control systems: From traditional to modern approaches. *Journal of Cybersecurity Research*, 15(4):34–45, 2020.

[14] Y. C. Chang and T. H. Lee. Advanced features enabled by ip and digital surveillance cameras. *IEEE Transactions on Industrial Electronics*, 68(11):9967–9975, 2021.

[15] Angelos Chatzimichail, Christos Chatzigeorgiou, Athina Tsanousa, Dimos Ntioudis, Georgios Meditskos, Fotis Andritsopoulos, Christina Karaberi, Panagiotis Kasnesis, Dimitrios G Kogias, Georgios Gorgogetas, et al. Internet of things infrastructure for security and safety in public places. *Information*, 10(11):333, 2019.

[16] David D Clark and David R Wilson. A comparison of commercial and military computer security policies. In *1987 IEEE Symposium on Security and Privacy*, pages 184–184. IEEE, 1987.

[17] Site: thorteaches.com.

[18] Antonio Carlos Cob-Parro, Cristina Losada-Gutiérrez, Marta Marrón-Romera, Alfredo Gardel-Vicente, and Ignacio Bravo-Muñoz. Smart video surveillance system based on edge computing. *Sensors*, 21(9):2958, 2021.

[19] Deepti Dhingra and Mohit Dua. The architecture of a video surveillance system. Scientific diagram accompanying the article "A novel chaotic map-based encryption scheme for surveillance videos", 2023. Accessed: 2025-06-05; figure downloaded from ResearchGate.

[20] Marwan Dhuheir, Abdullatif Albaseer, Emna Baccour, Aiman Erbad, Mohamed Abdallah, and Mounir Hamdi. Emotion recognition for healthcare surveillance systems using neural networks: A survey. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*, pages 681–687. IEEE, 2021.

[21] RACHID DJERBI. Formalisation et implantation d'une politique de securite dans le systÈme focalize. Magister's thesis, UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE, 2025.

[22] Dipti Anurag Doshi. A review on surveillance using thermal imaging system for animals and birds in the field of agriculture. *Int. J. Sci. Res. Sci. Technol*, 340, 2020.

[23] Mohamed El Beqqal and Mostafa Azizi. Review on security issues in rfid systems. *Advances in Science, Technology and Engineering Systems Journal*, 2(6):194–202, 2017. A review of vulnerabilities in RFID/keypad-based systems.

[24] Hakan Erdogmus, Massimo Morisio, and Tommi Männistö. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.

[25] What is smart surveillance and how can it protect your home? `https://www.espiamos.com/en/content/what-is-smart-surveillance-and-how-can-it-protect-your-home.html`. Accessed: 2025-05-18.

[26] Bouhamed et al. Maintainable aspect-oriented programming. 2025.

[27] Luciano Gavoni. Rfid exploitation and countermeasures. arXiv preprint arXiv:2110.00094, 2021. Discusses lack of formal threat models in RFID-based physical access systems.

[28] GeeksforGeeks. What is test-driven development (tdd)? Online article, 2023. Accessed: 2025-06-05; originally last updated December 4, 2023.

[29] Jayavardhana Gubbi, Rajkumar Buyya, S. Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[30] S. R. Haines and T. S. Malik. Architectural distinctions in cloud and iot-based access control systems. In *Proceedings of the International Conference on Cloud Computing and IoT*, pages 119–128. IEEE, 2019.

[31] Fadhil Hidayat, Ulva Elviani, Figo Agil Alunjati, and Muhammad Furqan Alfuady. Face recognition-based surveillance system in mining industry. *Jurnal Sistem Cerdas*, 7(2):226–236, 2024.

[32] Public mqtt broker - hivemq. `https://www.hivemq.com/mqtt/public-mqtt-broker/`. Accessed: 2025-05-18.

[33] What is mqtt? `https://www.hivemq.com/mqtt/`. Accessed: 2025-05-18.

[34] Weiming Hu, Wenhan Zheng, Longyin Xie, et al. Ai-based surveillance systems: Advances and challenges. *Pattern Recognition Letters*, 130:1–8, 2020.

[35] Ian Hutchinson and Michael Foster. From reactive to proactive: The paradigm shift in surveillance systems. *Journal of Security and Privacy*, 3(2):43–56, 2020.

[36] David S. Janzen and Hossein Saiedian. Does test-driven development really improve software design quality? *IEEE Software*, 25(2):77–84, 2008.

[37] Md Humaun Kabir, Sujit Roy, Md Tofail Ahmed, and Mahmudul Alam. Smart attendance and leave management system using fingerprint recognition for students and employees in academic institute. *International Journal of Scientific & Technology Research*, 10(6):268–276, 2021.

[38] P Lavanya, RIV Subba, V Selvakumar, and Shreesh V Deshpande. An intelligent health surveillance system: Predictive modeling of cardiovascular parameters through machine learning algorithms using lora communication and internet of medical things (iomt). *Journal of Internet Services and Information Security*, 14(1):165–179, 2024.

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[40] Gurusha Lulla, Abhinav Kumar, Govind Pole, and Gopal Deshmukh. Iot based smart security and surveillance system. In *2021 international conference on emerging smart computing and informatics (ESCI)*, pages 385–390. IEEE, 2021.

[41] Leszek Madeyski. Impact of test-driven development on software quality: a systematic literature review. *IET Software*, 4(5):343–358, 2010.

[42] Simo Mäkinen and Jürgen Münch. Effects of test-driven development: A comparative analysis of empirical studies. *Empirical Software Engineering*, 17(6):617–640, 2012.

[43] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[44] Andrea Mazzoleni, Andrea Rizzi, and Alberto Grasso. Smart surveillance: Advantages and challenges of integrating ai into video monitoring systems. *IEEE Access*, 8:175832–175850, 2020.

[45] Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007.

[46] Eclipse mosquitto - an open source mqtt broker. `https://mosquitto.org/`. Accessed: 2025-05-18.

[47] Selamat Muslimin, Yudi Wijanarko, Lucky Indra Kesuma, Renny Maulidda, Yordan Hasan, Hasan Basri, et al. Biometric fingerprint implementation for presence checking and room access control system. In *4th Forum in Research, Science, and Technology (FIRST-T1-T2-2020)*, pages 490–494. Atlantis Press, 2021.

[48] Mangai Natarajan. Cctv surveillance and crime prevention: A review of recent studies. *Security Journal*, 29(4):512–531, 2016.

[49] Elizabeth Palermo. Cctv: Digital eyes on the street. *IEEE Spectrum*, 42(11):30–34, 2005.

[50] Neha Patil, Shrikant Ambatkar, and Sandeep Kakde. Iot based smart surveillance security system using raspberry pi. In *2017 international conference on communication and signal processing (ICCSP)*, pages 0344–0348. IEEE, 2017.

[51] Eldho Paul, MS Kalepha, T Naveenkumar, and Mugeshbabu Arulmani. Auto surveillance using iot. In *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2022, Volume 3*, pages 407–415. Springer, 2022.

[52] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[53] Wei Ren and Stephen Barrett. Test-driven development, engagement in activity, and maintainability: An empirical study. *IET Software*, 17(3):181–190, 2023.

[54] [ResearchGate authors]. Embedded based tailgating/piggybacking detection security system. International Conference on Embedded Systems, 2014. Proposes camera-based system to detect tailgating in secure facilities.

[55] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[56] Sabab Ali Shah, Ghulam Mustafa Lakho, Hareef Ahmed Keerio, Muhammad Nouman Sattar, Gulzar Hussain, Mujahid Mehdi, Rahim Bux Vistro, Eman A Mahmoud, and Hosam O Elansary. Application of drone surveillance for advance agriculture monitoring by android application using convolution neural network. *Agronomy*, 13(7):1764, 2023.

[57] Himani Sharma and Navdeep Kanwal. Smart surveillance using iot: a review. *Radioelectronic and Computer Systems*, 2024(1):116–126, 2024.

[58] Sara Sicari, Antonio Rizzardi, L. A. Grieco, G. Boggia, M. Nitti, and P. Castoldi. A survey of security and privacy issues in internet of things. *Computer Networks*, 76:92–115, 2015.

[59] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6479–6488, 2018.

[60] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[61] Segundo Moisés Toapanta Toapanta, José Antonio Orizaga Trejo, and Luis Enrique Mafla Gallegos. Analysis of model clark wilson to adopt to the database of the civil registry of ecuador. In *Proceedings of the 21st conference of the Open Innovations Association FRUCT, Helsinki, Finland*, pages 6–10, 2017.

[62] Qiang Wang, Lei Qin, and Zhiqiang Chen. Intelligent surveillance systems: A review. *IEEE Transactions on Industrial Informatics*, 9(1):22–28, 2013.

[63] World Intellectual Property Organization (WIPO). General smart surveillance system. WIPO Patent Application IN389947136, 2021. Accessed: 2025-06-10; applicant: WIPO.

[64] World Intellectual Property Organization (WIPO). Iot-based smart surveillance system for theft detection and alert generation. WIPO Patent Application IN346888279, 2021. Accessed: 2025-06-10; applicant: WIPO.

[65] World Intellectual Property Organization (WIPO). Design of a smart security and surveillance system. WIPO Patent Application IN414836750, 2023. Accessed: 2025-06-10; applicant: WIPO.

[66] World Intellectual Property Organization (WIPO). Smart cctv camera system. WIPO Patent Application IN414845202, 2023. Accessed: 2025-06-10; applicant: WIPO.

[67] Yu Zeng, Wei Liu, and Changxin Li. The benefits and applications of digital/ip cameras in surveillance systems. *Journal of Computer Science and Technology*, 34(2):156–169, 2019.

[68] L. M. Zhang and H. C. Li. Modern access control systems: Benefits of scalability, flexibility, and security. *IEEE Transactions on Industrial Informatics*, 17(7):5069–5078, 2021.

[69] Wei Zhang and Jianhua Yang. *Networked Video Surveillance Systems*. Springer, 2015.

[70] Rui Zhao, Weiqiang Yan, and Yibing Wang. Deep learning-based smart surveillance systems: A review. *Artificial Intelligence Review*, 53:1–23, 2020.

[71] Zeming Zhao, Ping Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[72] Xinkun Zhou, Jiayuan Wang, Junyu Kang, and Zhifeng Zhang. Crowd counting via scale-adaptive convolutional neural network. *Pattern Recognition Letters*, 101:43–51, 2018.