



PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
Mohamed Khider University – BISKRA
Faculty of Exact Sciences
Department of Computer Science

Order No:M2/2025

Dissertation

Submitted for the academic Master's degree in Computer Science

Computer Science

Option: Information and Communication Networks and Technologies

Secure Algerian Smart Buildings

By:
Garti Anfal
Tahraoui Asma
Guettella Omayma

Defended on // 2025, in front of the jury composed of:

NOM PRENOM	Grade	President
Sahraoui Somia	MCA	supervisor
NOM PRENOM	Grade	Examiner

Academic year : 2024-2025

Acknowledgements

In the name of Allah, the Most Gracious, the Most Merciful.

All praise is due to Allah, who granted us the strength, patience, and perseverance to complete this work.

We would like to express our sincere gratitude to our supervisor, **Dr. Sahraoui Somia**, for her continuous support, insightful guidance, and valuable expertise throughout the course of this project. Her encouragement and constructive feedback have been instrumental in shaping our research and ensuring its successful completion.

We are also deeply thankful to the **honorable members of the jury** for their time, effort, and constructive evaluation of our work.

Our heartfelt thanks go to our **families**, whose unconditional love, encouragement, and support have been a constant source of strength throughout this journey.

Finally, we extend our gratitude to **all those who have contributed** to our academic and personal growth throughout this journey. This project represents the collective effort of our team and the guidance of those who supported us along the way.

Abstract

Smart buildings are increasingly adopting intelligent systems to enhance security, efficiency, and user experience. One of the critical components in these environments is access controlthe process of managing and regulating entry to various areas based on predefined rules and roles. However, traditional access control systems often suffer from issues related to centralization, lack of transparency, and limited scalability.

This project explores the use of blockchain technology, specifically Hyperledger, to implement a decentralized and secure access control system for smart buildings. The proposed solution aims to ensure that all access events are transparently recorded, permissions are managed dynamically, and only authorized entities are granted entry. By leveraging distributed ledger capabilities, the system provides a tamper-resistant and auditable platform for managing complex access scenarios involving residents, visitors, emergency personnel, and service workers.

The outcome of this work demonstrates the potential of integrating blockchain with smart environments to achieve a more secure, reliable, and efficient access control model.

Keywords- *Smart Building, Access Control, Blockchain, Hyperledger, Security, Decentralized Systems*

Abstraite

Les bâtiments intelligents adoptent de plus en plus de systèmes intelligents pour améliorer la sécurité, l'efficacité et l'expérience utilisateur. L'un des éléments essentiels de ces environnements est le contrôle d'accès : le processus de gestion et de régulation des accès aux différentes zones selon des règles et des rôles prédéfinis. Cependant, les systèmes de contrôle d'accès traditionnels souffrent souvent de problèmes de centralisation, de manque de transparence et d'évolutivité limitée. Ce projet explore l'utilisation de la technologie blockchain, et plus particulièrement d'Hyperledger, pour mettre en uvre un système de contrôle d'accès décentralisé et sécurisé pour les bâtiments intelligents. La solution proposée vise à garantir l'enregistrement transparent de tous les accès, la gestion dynamique des autorisations et l'accès réservé aux seules personnes autorisées. Grâce aux fonctionnalités d'un registre distribué, le système offre une plateforme inviolable et vérifiable pour gérer des scénarios d'accès complexes impliquant résidents, visiteurs, équipes d'urgence et agents de service.

Le résultat de ce travail démontre le potentiel de l'intégration de la blockchain avec des environnements intelligents pour obtenir un modèle de contrôle d'accès plus sûr, fiable et efficace.

Mots-clés- *Bâtiment intelligent, contrôle d'accès, blockchain, Hyperledger, sécurité, systèmes décentralisés*

الملخص

تعتمد المباني الذكية بشكل متزايد على الأنظمة الذكية لتعزيز الأمن والكفاءة وتجربة المستخدم. ومن أهم مكونات هذه البيئات التحكم في الوصول، وهو عملية إدارة وتنظيم الدخول إلى مختلف المناطق بناءً على قواعد وأدوار محددة مسبقاً. ومع ذلك، غالباً ما تعاني أنظمة التحكم في الوصول التقليدية من مشكلات تتعلق بالمركزية، وانعدام الشفافية، ومحدودية قابلية التوسع.

يستكشف هذا المشروع استخدام تقنية البلوك تشين، وتحديداً هايبرليدجر، لتطبيق نظام تحكم في الوصول لامركزي وآمن للمباني الذكية. يهدف الحل المقترح إلى ضمان تسجيل جميع أحداث الوصول بشفافية، وإدارة الأذونات ديناميكياً، ومنح الدخول للجهات المصرح لها فقط. ومن خلال الاستفادة من إمكانيات دفتر الأستاذ الموزع، يوفر النظام منصة مقاومة للتلاعب وقابلة للتدقيق لإدارة سيناريوهات الوصول المعقدة التي تشمل السكان والزوار وموظفي الطوارئ وعمال الخدمات. تُظهر نتائج هذا العمل إمكانية دمج البلوك تشين مع البيئات الذكية لتحقيق نموذج تحكم في الوصول أكثر أماناً وموثوقية وكفاءة.

الكلمات المفتاحية- المباني الذكية، التحكم في الوصول، بلوك تشين، هايبرليدجر، الأمن، الأنظمة اللامركزية

Contents

Abstract	ii
List of Figures	viii
List of Tables	x
General Introduction	1
1 Presentation of Smart Building Application	3
1.1 Introduction	4
1.2 Definition and Architecture of Smart Buildings	4
1.2.1 smart building	4
1.2.2 Key Components of Smart Buildings	5
1.2.3 Communication and Data Flow in Smart Building Infrastructure .	6
1.3 Functionalities and Objectives of Smart Buildings	8
1.3.1 Energy Efficiency and Sustainability	8
1.3.2 User Comfort and Convenience	8
1.3.3 Operational Cost Reduction	8
1.3.4 Safety and Security Measures	9
1.4 Role of Access Control in Smart Buildings	9
1.4.1 Definition of access control in Smart Buildings	9
1.4.2 Purpose of Access Control in Smart Buildings	9
1.4.3 How smart buildings manage access dynamically	10
1.5 Challenges in Modern Access Control	12
1.5.1 Managing different user types	12
1.5.2 Scalability and Flexibility Issues	13
1.5.3 Real-Time Monitoring and Response Needs	13
1.5.4 Integration with Other Building Systems	14

1.6	Conclusion	14
2	Overview on Blockchain technology	15
2.1	Introduction	16
2.2	Blockchain Technology	16
2.2.1	Définition	16
2.2.2	Core Components	16
2.2.3	Key Features	17
2.2.4	security features of Blockchain	18
2.2.5	How Blockchain Works?	20
2.2.6	Blockchain Types	22
2.2.7	What are blockchain protocols?	24
2.3	Hyperledger fabric	25
2.3.1	Définition	25
2.3.2	Key Features	25
2.3.3	Key Components of Hyperledger Fabric	28
2.3.4	How organizations collaborate in a fabric network ?	29
2.3.5	Benefits of Hyperledger fabric	31
2.3.6	Use Cases	31
2.4	State of the art	32
2.4.1	Bindra et al. (2020) Flexible, Decentralized Access Control for Smart Buildings with Smart Contracts	32
2.4.2	NineSmart Smart Access QR Code Access Control	33
2.4.3	SPCL: A Smart Access Control System That Supports Blockchain	33
2.5	Comparison table	33
2.6	Conclusion	34
3	Proposed solution	35
3.1	Introduction	36
3.2	Use Case diagram	36
3.3	Global architecture of the system	38
3.4	Detailed architecture of the system	39
3.4.1	Hyperledger Fabric Network	39
3.5	Hyperledger Fabric Network Setup and Development Steps	42
3.6	Deploy Chaincode(Smart Contract)	43

3.7	Application Programming Interface(Api)	44
3.8	Functionality of the system	45
3.8.1	Sequence diagram of the system	45
3.8.2	Sequence diagram (Hlf)	49
3.9	Conclusion	60
4	Development and evaluation	61
4.1	Introduction	62
4.2	Software tools	62
4.3	Hardware Tools	64
4.4	Results and discussion	69
4.5	Implementation of the System	74
4.5.1	Login	74
4.5.2	Dashboard of Admin	75
4.5.3	Resident Dashboard	80
4.5.4	Scan Page	82
4.5.5	Guide	83
4.6	Conclusion	84
	General Conclusion	85
	Appendix: Smart Contracts	92

List of Figures

1.1	Smart Building Overview	4
1.2	Illustration of Occupancy and Vacancy Sensors	5
1.3	Illustration of Smart Building Sensors	5
1.4	Automated building subsystems and IT context	7
1.5	Attribute-based access control example	11
1.6	Occupant-Centric Building Controls example	12
2.1	Components of Blockchain	17
2.2	Cryptographic hashing	19
2.3	Cryptographic hashing	19
2.4	Consensus mechanisms	20
2.5	Smart Contract	20
2.6	How Blockchain Work	22
2.7	public Blockchain	23
2.8	Private blockchain	24
2.9	NineSmart	33
3.1	Use Case diagram	37
3.2	Global Architecture	39
3.3	Deatiled architecture of the system	41
3.4	Development Steps of Hlf	42
3.5	Deploy Chainecode Steps	43
3.6	Api	44
3.7	Sequence diagram	46
3.8	Add Residents to stock in ledger	49
3.9	Update Resident Informations	50
3.10	Block/Unblock Resident	51
3.11	Add Visitor	52

3.12 Block/Unblock Visitor	54
3.13 Update Visitor	56
3.14 Add Visit Request	57
3.15 approve/reject request of visitor	59
4.1 Diagram illustrating the typical hardware setup for NFC-based access control.	65
4.2 Chaincode Metrics for Peer Requests Over Time	70
4.3 Ledger Blockchain Height Metric for Resident Peer	71
4.4 StateDB Commit Time Metric for Peers on <code>residentschannel</code>	72
4.5 ledger block processing time	73
4.6 Login Interface	75
4.7 Dashboard Overview	76
4.8 Manage Appartments	76
4.9 Manage Residents	77
4.10 Manage Requests	78
4.11 Exemple the guide in Manage Residents page	84

List of Tables

2.1	Comparative Summary of Access Control Solutions	33
3.1	Distribution of Peers and their Responsibilities	40
3.2	MSP Assignment and Roles	41
4.1	Hardware Components-1	66
4.2	Hardware Components-2	67
4.3	Hardware Components-3	68
4.4	Hardware Components-4	69

General Introduction

1-Context

The rapid advancement of IoT (Internet of Things) in general [1] and smart building technologies in particular [2] have led to the integration of various systems aimed at enhancing operational efficiency and occupant experience. Central to these systems is access control, which governs the entry and movement of individuals within the building. Traditional access control methods often rely on centralized databases and static permissions, which can be susceptible to security breaches and administrative errors. With the increasing complexity of building operations and the need for real-time access management, there is a growing demand for more secure, transparent, and efficient access control solutions.[3].

2-Problematic

Existing access control systems in smart buildings face several challenges:

- **Centralization:** Relying on a single point of control increases vulnerability to attacks and system failures.
- **Lack of Transparency:** Difficulty in auditing access events and permissions can lead to accountability issues.
- **Scalability Issues:** As buildings expand or integrate with other systems, traditional methods may struggle to manage the increased complexity.

These limitations necessitate the exploration of decentralized technologies that can offer enhanced security and flexibility.

3-Objectives

This master thesis aims to design and implement a blockchain-based access control system using Hyperledger Fabric to address the aforementioned challenges. The specific objectives are:

- ◆ **Decentralization:** Eliminate single points of failure by distributing control across multiple nodes.

- ◆ **Transparency:** Enable immutable logging of access events for auditability.
- ◆ **Scalability:** Ensure the system can handle a growing number of users and access points without degradation in performance.
- ◆ **Role-Based Access Control (RBAC)** Implement fine-grained access policies based on user roles and attributes.

4-Thesis Structure

The current report is structured into four chapters:

- ◆ **The 1st Chapter:** Provides an overview of smart building applications, highlighting the importance of efficient access control systems.
- ◆ **The 2nd Chapter:** Offers a comprehensive review of blockchain technology, with a focus on Hyperledger Fabric, and its applicability to access control systems.
- ◆ **The 3rd Chapter:** Presents the proposed blockchain-based access control solution, detailing its architecture, components, and functionalities.
- ◆ **The 4th Chapter:** Describes the development process, implementation challenges, and evaluation of the system's performance and effectiveness.

Chapter 1

Presentation of Smart Building Application

1.1 Introduction

A smart building uses advanced technologies like sensors, actuators, and IoT devices to automate and optimize functions such as lighting, HVAC, energy use, and security, enhancing comfort, safety, and efficiency. It relies on real-time data and intelligent systems to adapt to environmental changes and user needs. A key feature is access control, which restricts entry to authorized individuals, safeguarding physical and digital assets.

This chapter introduces smart buildings, highlights the role of access control, and explains why modern solutions are needed to overcome the limits of traditional systems.

1.2 Definition and Architecture of Smart Buildings

A smart building is a structure equipped with advanced systems and technologies, such as IoT devices, sensors, and automation, to monitor and control building operations like heating, ventilation, lighting, and security. It aims to enhance energy efficiency, occupant comfort, and operational performance. Through intelligent data collection and analysis, smart buildings adapt to user needs and optimize resource usage in real time.

1.2.1 smart building

A smart building is defined as a structure equipped with advanced technologies that facilitate automated processes to control and manage building operations. These operations encompass HVAC, lighting, security, and other systems, all of which respond dynamically to environmental conditions and occupant needs . [4].



Figure 1.1: Smart Building Overview [5].

1.2.2 Key Components of Smart Buildings

Smart buildings integrate advanced technologies to enhance operational efficiency, occupant comfort, and sustainability. The primary components include:

1.2.2.1 Sensors and IoT Devices

Sensors collect real-time data on environmental parameters such as temperature, humidity, occupancy, and light levels. IoT devices facilitate communication between these sensors and central systems, enabling automated responses.

- Occupancy sensors detect the presence of individuals to adjust lighting and HVAC systems accordingly [6].

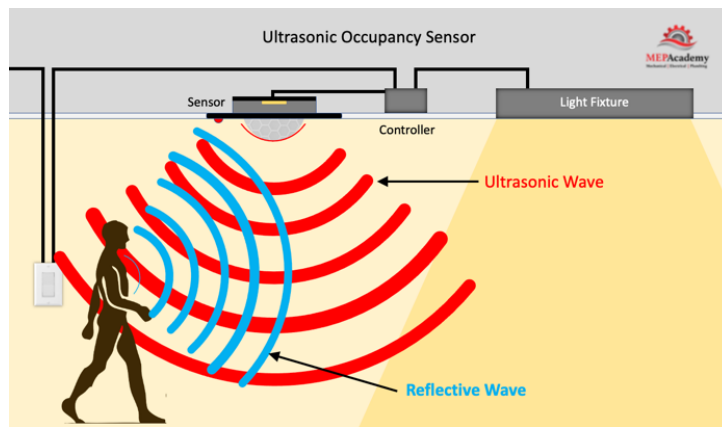


Figure 1.2: Illustration of Occupancy and Vacancy Sensors [7].

- IoT devices, including smart thermostats and lighting systems, allow for remote monitoring and control.



Figure 1.3: Illustration of Smart Building Sensors [8].

1.2.2.2 Building Automation Systems (BAS)

Building Automation Systems (BAS) serve as the central hub, integrating various building systems such as HVAC, lighting, and security. They process data from sensors to optimize

building operations.

- BAS can maintain indoor climates within specified ranges and provide alerts for system malfunctions.

1.2.2.3 Building Energy Management Systems (BEMS)

Building Energy Management Systems (BEMS) are integrated, computerised systems designed to monitor, control, and optimise energy-related building services such as heating, ventilation, air conditioning (HVAC), lighting, and power systems. They play a pivotal role in enhancing energy efficiency, reducing operational costs, and supporting environmental sustainability within smart buildings.

- EMS can coordinate energy usage across multiple systems, including lighting and HVAC [9].

1.2.2.4 Communication Networks

Robust communication networks ensure seamless data transmission between devices and systems. They support various protocols and standards to facilitate interoperability.

- Modern systems utilize protocols like BACnet and KNX for device communication.

1.2.2.5 User Interfaces

User interfaces, such as mobile apps and dashboards, allow occupants and facility managers to interact with building systems, monitor performance, and make adjustments as needed.

1.2.3 Communication and Data Flow in Smart Building Infrastructure

1.2.3.1 Network Infrastructure

Smart buildings utilize a hybrid network infrastructure combining both wired and wireless technologies to connect devices and systems [10]

- **Wired Networks:** Technologies like Ethernet provide reliable, high-bandwidth connections essential for data-intensive applications.
- **Wireless Networks:** Protocols such as Wi-Fi, Zigbee, and Bluetooth offer flexibility and scalability, allowing easy integration of IoT devices without extensive cabling.

1.2.3.2 Communication Protocols

To ensure interoperability among diverse devices, smart buildings utilize standardized communication protocols.

- **BACnet:** Facilitates communication among building automation and control systems. [11]
- **MQTT (Message Queuing Telemetry Transport):** A lightweight protocol ideal for devices with limited bandwidth. [10]
- **CoAP (Constrained Application Protocol):** Designed for simple electronics to communicate over the internet. [10]

1.2.3.3 Data Acquisition and Processing

Sensors and IoT devices collect vast amounts of data, which are then transmitted to centralized systems for processing. This data includes information on temperature, occupancy, lighting levels, and energy consumption. [12]

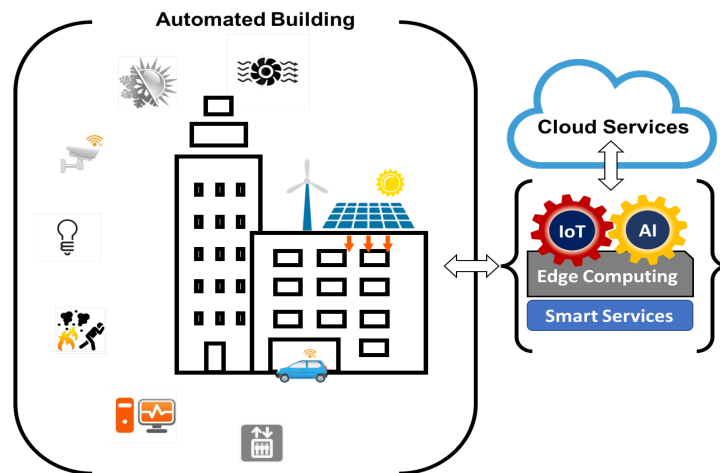


Figure 1.4: Automated building subsystems and information technologies context [13].

- **Edge Computing:** Processes data locally at the device level, reducing latency and bandwidth usage.
- **Cloud Computing:** Stores and analyzes data on remote servers, enabling advanced analytics and remote access.

1.2.3.4 Building Management Systems (BMS)

The BMS acts as the central hub, integrating data from various subsystems (HVAC, lighting, security) and enabling coordinated control strategies. It ensures that all systems operate cohesively, enhancing efficiency and occupant comfort. Advanced BMS platforms support real-time monitoring, predictive maintenance, and energy optimization, contributing to the overall sustainability and operational excellence of smart buildings. [14]

1.3 Functionalities and Objectives of Smart Buildings

Smart buildings leverage advanced technologies to enhance energy efficiency, occupant comfort, operational efficiency, and safety. Below are the primary functionalities and objectives:

1.3.1 Energy Efficiency and Sustainability

- **Automated Energy Management:** Smart buildings utilize intelligent systems to monitor and control energy consumption, optimizing the use of lighting, HVAC, and other systems to reduce energy waste
- **Integration of Renewable Energy:** Incorporating renewable energy sources, such as solar panels, allows smart buildings to reduce reliance on non-renewable energy and decrease their carbon footprint.

1.3.2 User Comfort and Convenience

- **Enhanced Comfort through Active Structures and Home Automation:** Smart buildings incorporate active structures and home automation systems that adapt to occupants' needs, improving overall comfort levels. These technologies adjust environmental conditions such as lighting and temperature in real-time, responding to user preferences and activities. [15]
- **Well-being through Ambient Assisted Living Technologies:** The integration of ambient assisted living (AAL) technologies in smart buildings supports the well-being of occupants, particularly the elderly and individuals with special needs. These systems provide assistance in daily activities, health monitoring, and emergency responses, enabling independent living and enhancing quality of life. [15]

1.3.3 Operational Cost Reduction

- **Predictive Maintenance:** Implementing predictive maintenance strategies in smart buildings can significantly reduce operational costs. Real-time monitoring of equipment performance allows for early detection of potential failures, minimizing downtime and repair expenses. Studies indicate that predictive maintenance can lower maintenance costs by 25--30% and reduce equipment downtime by 35--45% [16].
- **Optimized Resource Management:** The integration of Internet of Things (IoT) technologies enables efficient resource management in smart buildings. IoT sensors provide real-time data on energy consumption, allowing for dynamic adjustments to systems like HVAC and lighting. This approach can lead to energy consumption reductions of up to 30% and operational cost savings of 20%. [17]

1.3.4 Safety and Security Measures

- **Integrated Security Systems:** Smart buildings employ advanced security measures, including surveillance cameras, access control, and intrusion detection systems, to ensure occupant safety. [18].
- **Emergency Response Automation:** TIn the event of emergencies, such as fires or gas leaks, smart systems can automatically trigger alarms, unlock exits, and notify emergency services. [19].

1.4 Role of Access Control in Smart Buildings

1.4.1 Definition of access control in Smart Buildings

Access control in smart buildings refers to the integration of advanced security systems that regulate and monitor entry to various areas within a building, ensuring that only authorized individuals have access to specific zones or resources. These systems are pivotal in enhancing the safety, efficiency, and user experience within intelligent infrastructures.

In smart buildings, access control systems are not isolated; they are interconnected with other building management systems such as HVAC, lighting, and surveillance. This integration allows for dynamic responses to various scenarios. For instance, during emergencies, the access control system can automatically unlock doors for evacuation or restrict access to hazardous areas, thereby ensuring occupant safety and facilitating efficient emergency responses.

Moreover, modern access control systems in smart buildings leverage technologies like biometric authentication, mobile credentials, and cloud-based management. These advancements not only bolster security but also provide flexibility and convenience to users, allowing for seamless access experiences and efficient administrative control over access permissions. [20].

1.4.2 Purpose of Access Control in Smart Buildings

1.4.2.1 Enhancing Security and Preventing Unauthorized Access

Access control systems are fundamental in safeguarding smart buildings against unauthorized entry and potential threats. By regulating who can enter specific areas, these systems prevent unauthorized access to sensitive zones, thereby mitigating risks associated with theft, vandalism, or terrorism. Advanced access control mechanisms can also integrate with surveillance and intrusion detection systems to provide real-time threat assessment and response.

1.4.2.2 Facilitating Dynamic and Context-Aware Access Management

Smart buildings often require flexible access control that adapts to changing contexts, such as time of day, occupancy levels, or specific events. Dynamic access control sys-

tems can adjust permissions in real-time, ensuring that access rights align with current conditions and policies. This adaptability enhances both security and operational efficiency. [21].

1.4.2.3 Ensuring Data Privacy and Compliance

With the increasing digitization of building operations, protecting occupant data has become paramount. Access control systems contribute to data privacy by ensuring that only authorized personnel can access sensitive information and areas. Moreover, these systems help organizations comply with data protection regulations by providing audit trails and enforcing strict access policies. [22].

1.4.2.4 Improving Operational Efficiency and Resource Optimization

By automating entry and exit processes, access control systems reduce the need for manual monitoring and staffing, leading to cost savings. Additionally, data collected from access logs can inform building usage patterns, enabling better resource allocation and energy management. [3].

1.4.2.5 Enhancing User Experience and Convenience

Modern access control solutions offer convenience to occupants through features like mobile credentials, biometric authentication, and personalized access rights. These technologies not only streamline entry processes but also enhance the overall user experience by providing seamless and secure access to building amenities. [3].

1.4.3 How smart buildings manage access dynamically

Smart buildings dynamically manage access by integrating advanced technologies that adapt to real-time conditions, user behaviors, and contextual factors. This dynamic access control enhances security, operational efficiency, and user experience. Below are key mechanisms through which smart buildings achieve dynamic access management:

1.4.3.1 Attribute-Based Access Control (ABAC)

ABAC is a method of implementing access control policies that is highly adaptable and can be customized using a wide range of attributes, making it suitable for use in distributed or rapidly changing environments. ABAC determines access rights based on a combination of attributes related to the user (subject), the resource (object), the action, and the environment. This model allows for fine-grained and context-aware access decisions, accommodating complex policies that consider factors like time, location, and user role [23].

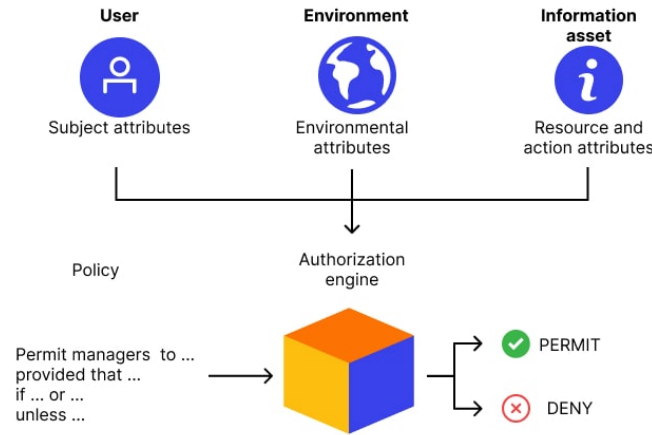


Figure 1.5: Attribute-based access control example [24].

1.4.3.2 Integration with Building Management Systems (BMS)

Modern access control systems are integrated with BMS, enabling centralized monitoring and control of various building functions such as HVAC, lighting, and security. This integration allows for coordinated responses to access events, such as adjusting environmental settings based on occupancy. The integration becomes possible due to the applications of advanced IT technologies such as Web Services. [25]

1.4.3.3 AI-Powered Access Control Systems

Artificial Intelligence enhances access control by enabling systems to learn from patterns, predict potential security threats, and adapt to new scenarios without manual intervention. AI algorithms can analyze vast amounts of data to identify anomalies and make informed access decisions. AI-powered smart building systems can use more sophisticated generative AI (GenAI) training models for well-informed insight and optimization, or they can use conventional AI technology to evaluate data, generate predictions, and carry out activities. [26]

1.4.3.4 Blockchain-Based Access Control

A flexible and secure methodology has been developed to manage building access privileges for both long-term occupants and short-term visitors. This approach employs blockchain smart contracts to define, grant, audit, and revoke fine-grained permissions in a decentralized manner. By leveraging information from Brick and BOT models of the building, the system specifies smart contracts that facilitate this access control. An application scenario in a real office building demonstrates that this method can significantly reduce administrative overhead while providing detailed, auditable access control. [3]

1.4.3.5 Occupant-Centric Building Controls

Occupant-Centric Building Controls (OCC) use real-time data on occupant presence, preferences, and environmental conditions to automatically adjust building systems like HVAC and lighting. This dynamic control approach enhances user comfort and significantly improves energy efficiency. Despite their benefits, practical implementations face challenges such as sensor reliability, data integration, and privacy concerns. [27]

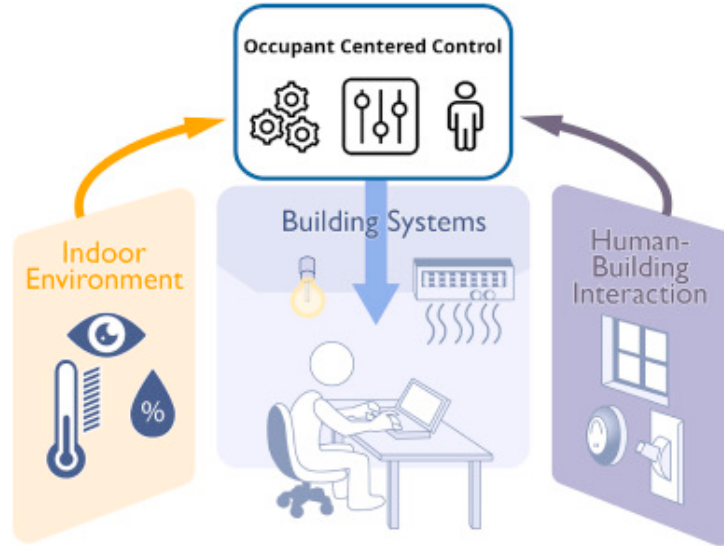


Figure 1.6: Occupant-Centric Building Controls example [27].

1.5 Challenges in Modern Access Control

As smart buildings become more complex and interconnected, access control systems face growing demands in functionality, scalability, and security. Modern systems must not only regulate physical entry but also adapt to diverse users, integrate with other building technologies, and respond in real time to events. This section highlights key challenges that affect the design and deployment of access control in smart environments.

1.5.1 Managing different user types

Modern smart buildings must manage access for various user types: employees, visitors, and service personnel, each with unique access needs. While employees often have fixed, role-based privileges, visitors receive limited, temporary access. This rigid approach lacks flexibility and requires constant updates and oversight. Although role- or attribute-based schemes offer more adaptable solutions, implementing and enforcing fine-grained access policies remains a significant challenge. [3]

- **Employees (Regular Occupants):** Typically granted persistent credentials and access according to their department or job role. For instance, a researcher might have 24/7 card access to lab areas, while a receptionist is limited to the lobby and

offices. Managing hundreds of such privileges (and changes, e.g. promotions or transfers) places a high administrative burden.

- **Visitors:** Usually require pre-registration and limited access (e.g. only public areas), often under escort. large commercial buildings, long-term occupants are given access privileges based on their roles, while visitors have to be escorted by their hosts. This approach is conservative and inflexible. Systems must simplify visitor check-in (e.g. kiosks, mobile credentials) while preventing unauthorized entry.
- **Service Personnel:** Maintenance workers or cleaning staff need specialized, time-bound access (e.g. to mechanical rooms or after-hours entry). Their schedules and duties are variable, so the control system must issue temporary or conditional credentials (for example, one-time codes) and revoke them automatically after tasks are done.

Overall, managing these distinct user types demands sophisticated access models. New research proposes using decentralized, fine-grained policies to automate granting and revoking permissions for both occupants and visitors.

1.5.2 Scalability and Flexibility Issues

Smart buildings host thousands of users and devices, requiring access control systems to scale without performance loss. scalable architectures must support horizontal growth (e.g., new wings or doors) and leverage cloud or distributed systems to manage load efficiently. [28]

- **Scalable Architecture:** Systems should allow horizontal expansion and distributed control using open-source or modular components
- **Performance Under Load:** High transaction volumes during peak times (e.g., shift changes) require low-latency processing, supported by local decision-making or edge computing.
- **Flexibility for Change:** Adapting to new credentials (biometrics, mobile apps) or organizational updates should be seamless. Open standards and centralized policy updates aid this

Maintaining performance and security at scale demands robust design, including redundancy and failover, to prevent system failure as size and complexity increase

1.5.3 Real-Time Monitoring and Response Needs

Effective access control in smart buildings requires real-time event detection and rapid response to threats. Integrated with sensors, cameras, and alarms, systems must process events instantly e.g., triggering alerts or lockdowns when doors are forced open. [28]

- **Low-Latency Processing:** Access decisions must occur within milliseconds. edge and fog computing reduce latency by processing data locally, avoiding cloud delays.
- **Integrated Surveillance:** Real-time analysis of video feeds and sensor data (e.g., tailgating detection) enhances situational awareness. AI and streaming analytics help correlate access logs with surveillance.
- **Automated Response:** Systems should react automatically to anomalies e.g., locking areas or alerting security upon intrusion or cyber anomalies. This demands robust network infrastructure and event-handling logic.

In short, real-time performance is critical, requiring edge computing, intelligent analytics, and responsive architectures to maintain building security.

1.5.4 Integration with Other Building Systems

Modern access control systems often interface with HVAC, lighting, and elevator systems to enhance efficiency and security. For instance, badge data can trigger elevators to restrict floor access or inform HVAC systems to optimize energy usage. [28]

- **Interoperability Challenges:** Systems often use different protocols (e.g., BACnet, MQTT), making integration difficult. adopting standardized, modular architectures (e.g., OpenFog) simplifies integration and improves interoperability.
- **Coordinated Automation:** Integration enables smart behaviors like elevator access based on credentials or climate control based on occupancy. These require secure, real-time data exchange across systems.
- **Security Risks:** Integration expands the attack surface. A breach in one system (e.g., lighting) could impact access control. All interactions must be securely authenticated and follow strict policies.

In summary, integration boosts smart functionality but demands open standards, strong security, and careful system coordination

1.6 Conclusion

In this chapter, we introduced the concept of smart buildings and defined their architecture, key components, and core functionalities such as energy efficiency, user comfort, and security. We also explored the crucial role of access control systems and discussed the main challenges they face, including scalability, real-time response, and system integration. These foundational concepts are essential for understanding our proposed solution.

In the next chapter, we will provide an overview of blockchain technology, including its definitions, core components, and security features. We will also explore Hyperledger Fabric and examine how its structure and capabilities can support secure and decentralized systems in modern applications.

Chapter 2

Overview on Blockchain technology

2.1 Introduction

With the increasing reliance on smart systems in modern buildings, it has become essential to adopt advanced digital technologies to ensure security and transparency in access management. Blockchain is one of the most prominent of these technologies, as it provides a decentralized and secure framework for recording transactions and verifying permissions.

In this chapter, we present the fundamental concepts of blockchain technology, its main components, and how it works, with a focus on the security and transparency it offers. We also specifically address the Hyperledger Fabric platform, as it represents the optimal choice for implementing our project due to its support for private networks, identity management, and efficient execution of smart contracts.

2.2 Blockchain Technology

2.2.1 Définition

Blockchain is a decentralized technology that keeps a common, secure ledger through a network of computers (known as nodes or peers). Rather than depending on a central server, every transaction or modification is validated by a consensus of nodes and recorded in blocks that are protected by cryptography and can not be changed.[\[29\]](#)

2.2.2 Core Components

2.2.2.1 Blocks:

The basic building element of a blockchain is a block, which is an immutable record in the blockchain network made up of a collection of verified transactions and cryptographic connections to earlier blocks.[\[30\]](#)

2.2.2.2 Nodes:

Within a blockchain network, a blockchain node is a point of connection that receives, stores, transmits, and verifies data. A node is in charge of the network's security, decentralization, and consensus procedures. It also keeps a copy of the complete blockchain ledger, which guarantees the integrity of transactions and blocks.[\[31\]](#)

2.2.2.3 Transactions:

Transferring assets or data between participants and recording it on a decentralized digital ledger is known as a blockchain transaction. Because each transaction is verified by several parties and runs on a peer-to-peer network, blockchain transactions are impervious to manipulation.[\[32\]](#)

2.2.2.4 Miners (or Validators):

The task of proposing and certifying new blocks falls to specific nodes. Thus, the process of mining the blockchain creates new blocks. Miners compete to solve difficult cryptographic puzzles in order to add a new block to the blockchain. The winner is the first person to figure out the puzzle.

Through the verification and recording of transactions, mining preserves the blockchain's decentralized structure. Because of this, miners that are successful are rewarded with cryptocurrency. [32]

2.2.2.5 Consensus:

A blockchain system establishes rules about participant consent for recording transactions. You can record new transactions only when the majority of participants in the network give their consent.[33]

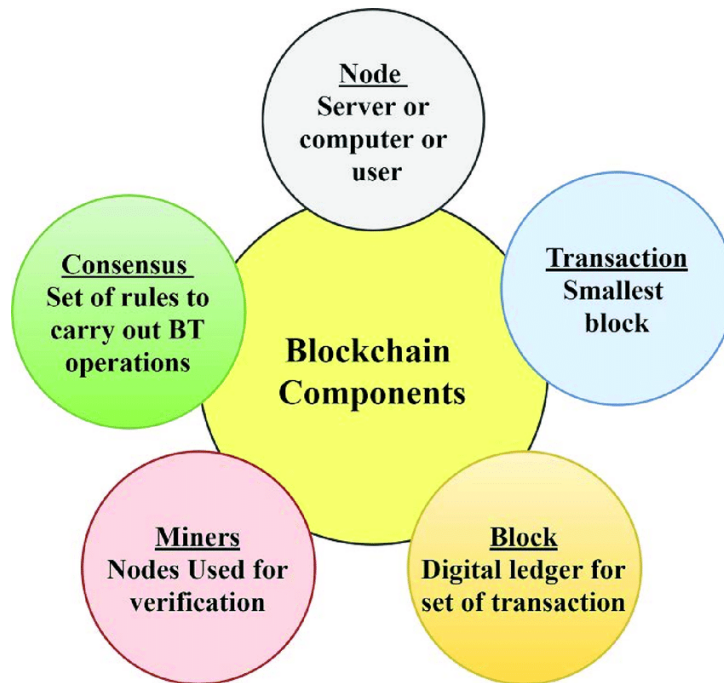


Figure 2.1: Components of Blockchain [34].

2.2.3 Key Features

2.2.3.1 Decentralization:

In blockchain, decentralization means moving authority and decision-making from a centralized entity (person, group, or organization) to a distributed network. Transparency is a technique used by decentralized blockchain networks to lessen member trust. Participants in these networks are also discouraged from using power or influence over one another in ways that impair the network's ability to function.[35]

2.2.3.2 Immutability:

Immutability is the inability to change or modify anything. Once a transaction has been entered into the shared ledger, no participant can alter it. In the event that a transaction record contains an error, both transactions are accessible to the network and must be reversed by adding a new transaction.[\[35\]](#)

2.2.3.3 Transparency:

Because blockchain technology is auditable and unchangeable, it offers transparency. A transparent and permanent record of all transactions or acts is created by the blockchain, which prevents data from being changed or removed once it is recorded. This increases accountability and confidence, which makes it perfect for use cases like voting systems and supply chain management where transparency is essential.[\[36\]](#)

2.2.3.4 Interoperability:

Blockchain could make it possible for various networks and systems to communicate with one another. A connected ecosystem is produced by the smooth integration and data exchange between various blockchains made possible by blockchain protocols and standards. This facilitates interoperable apps, cross-chain transactions, and improved stakeholder collaboration.[\[36\]](#)

2.2.4 security features of Blockchain

2.2.4.1 Cryptographic Hashing:

Blockchain protects data and transactions with cutting-edge encryption algorithms. Cryptographic hashes are used to connect each block in the blockchain to its predecessor.[\[36\]](#)

Thus, when input data (such as a transaction) is sent through a hash function, a fixed-size output known as the hash is produced. Each blockchain block contains the hash of the one before it in order to create a chain. This makes it impossible to change data once it has been recorded[\[32\]](#).

Furthermore, blockchain's distributed architecture increases security by making it resistant to single points of failure.[\[36\]](#)

How Hashing Works

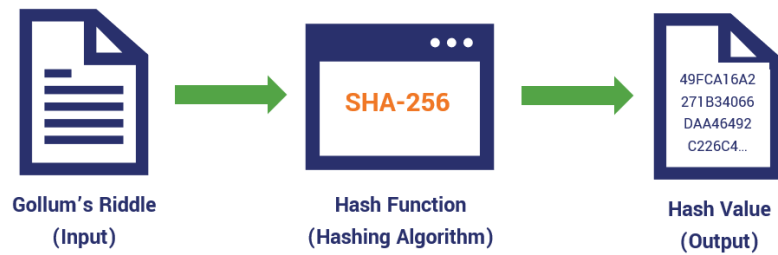


Figure 2.2: Cryptographic hashing [37]

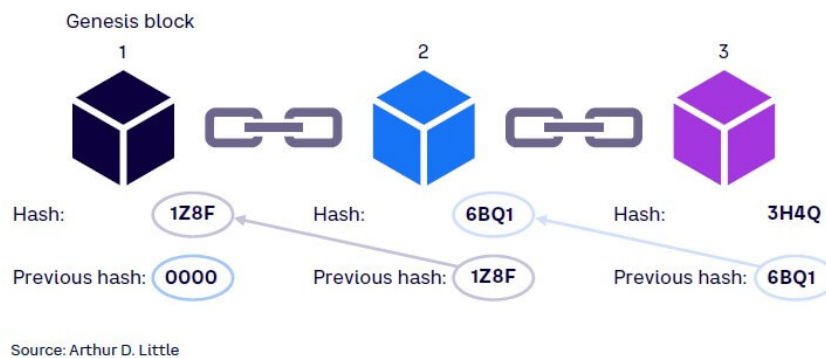


Figure 2.3: Cryptographic hashing [38]

2.2.4.2 Consensus Mechanisms:

Blockchain networks employ a consensus technique to get dispersed processes or systems to agree on a common data value. It guarantees that the network's nodes are all in sync and running the same version of the blockchain.[39]

- Types of consensus mechanisms include:

- **Proof of Work (PoW):** To verify and validate transactions, participants must resolve challenging mathematical puzzles.
- **Proof of Stake (PoS):** The quantity of coins that validators own and are prepared to "stake" as collateral determines their selection.
- **Delegated Proof of Stake (DPoS):** Stakeholders designate delegates to verify transactions on their behalf.
- **Practical Byzantine Fault Tolerance (PBFT):** enables nodes to come to an agreement even in the event that some nodes malfunction or behave maliciously.



Figure 2.4: Consensus mechanisms [40].

2.2.4.3 Smart Contract:

Smart contracts, or self-executing contracts with predetermined criteria that are automatically enforced when those requirements are met, are supported by blockchain technology. Smart contracts allow agreements to be executed automatically, transparently, and securely without the need for middlemen, which lowers costs and boosts productivity across a range of sectors, including supply chain management, real estate, and banking.[36]



Figure 2.5: Smart Contract [41].

2.2.5 How Blockchain Works?

Step 1 Record the Transaction

A blockchain transaction demonstrates the transfer of digital or tangible (physical) assets between participants in the network. It's recorded as a data block and can include details like the following:

- Who was involved in the transaction?
- What happened during the transaction?
- When did the transaction occur?
- Where did the transaction occur?
- Why did the transaction occur?
- How much of the asset was exchanged?
- How many pre-conditions were met during the transaction?[35]

Step 2 Gain Consensus

The majority of users (participants) on the distributed blockchain network have to concur that the transaction that was recorded is legitimate. The rules of agreement can change depending on the kind of network, but they are usually set up at the beginning.[35]

Step 3 Link the Blocks

The blockchain records transactions in blocks, which are comparable to the pages of a ledger book, when the participants have come to an agreement. The transactions are appended to the new block along with a cryptographic hash. The hash serves as a chain connecting the blocks.

Data tampering can be detected since the hash value changes if the block's contents are altered, whether on purpose or accidentally.

As a result, you are unable to alter the blocks and chains and they link securely. The blockchain as a whole and the prior block's verification are strengthened with each new block. This is comparable to building a skyscraper out of wooden blocks. Only blocks can be stacked on top of one another, and the tower collapses if a block is taken out of the center.[35]

Step 4 Share the Ledger

All network users receive the most recent version of the central ledger from the system.[35]

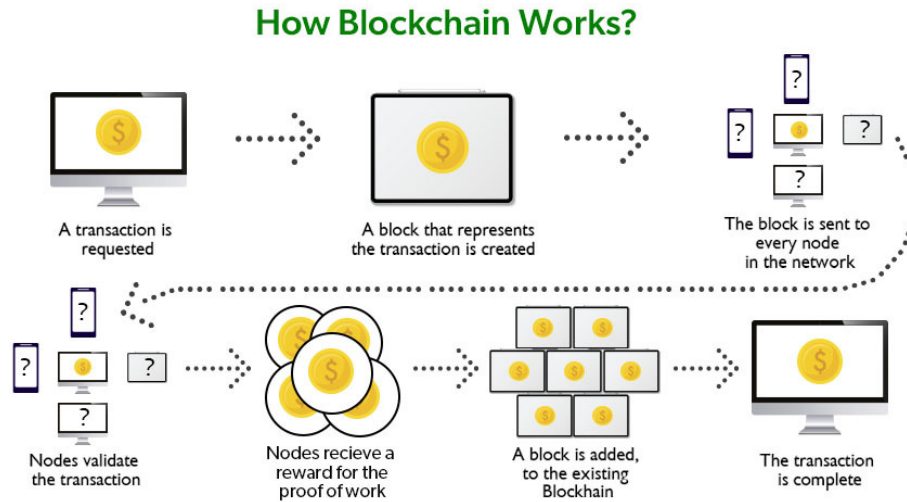


Figure 2.6: How Blockchain Work [42].

2.2.6 Blockchain Types

2.2.6.1 Public Blockchain:

A public blockchain is an idea in which anybody may sign up and participate in the main functions of the blockchain network. Anyone can view, publish, and audit the ongoing operations on a public blockchain network, which contributes to the decentralized, self-determining aspect that is frequently permitted when discussing blockchain. Data on a public blockchain is safe because, once verified, it cannot be altered.

The public blockchain is completely decentralized, has access to and control over the ledger, provides data that is always accessible, and has a central authority that oversees each block in the chain. All operations are conducted in a public manner. There is no need to obtain authorization in order to access the public blockchain because it is not being handled alone. In a network or chain, anyone can set their own node or block.

All of the blocks are connected in a peer-to-peer fashion once a node or block has established itself in the chain of blocks. If the block is attacked, a copy of the data is created, and only the block's original author can access it.[43]

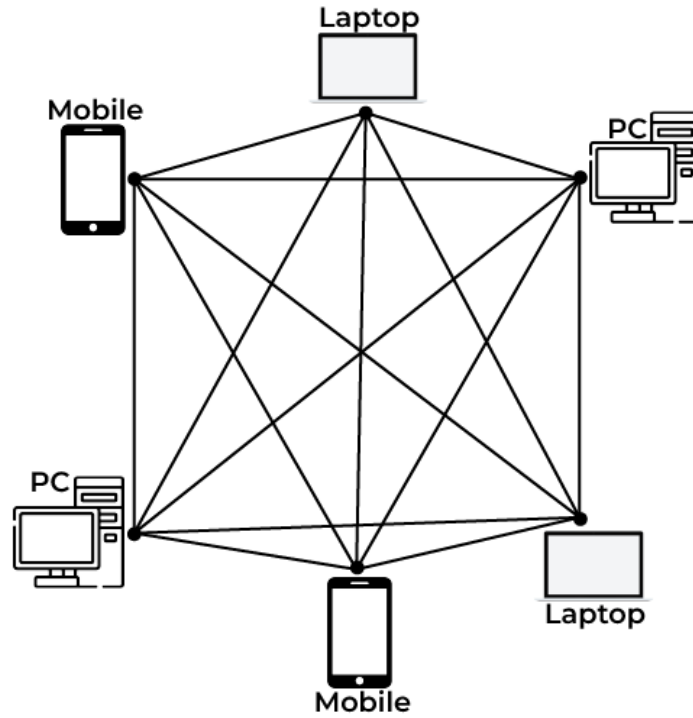


Figure 2.7: public Blockchain[43].

2.2.6.2 Private Blockchain:

To access a private blockchain, miners require authorization. Permissions and restrictions, which restrict network participation, are the foundation of its operation. The other stakeholders won't have access to a transaction; only the parties involved will be aware of it.

It is also known as a permission-based blockchain since it operates on permissions.

Unlike public blockchains, private blockchains are controlled by the network's owner.

The blockchain is managed by a trustworthy individual who will also manage the private chain network's access privileges and restrict who has access to the private blockchain.

There is a chance that access to the private blockchain network will be restricted.[43]

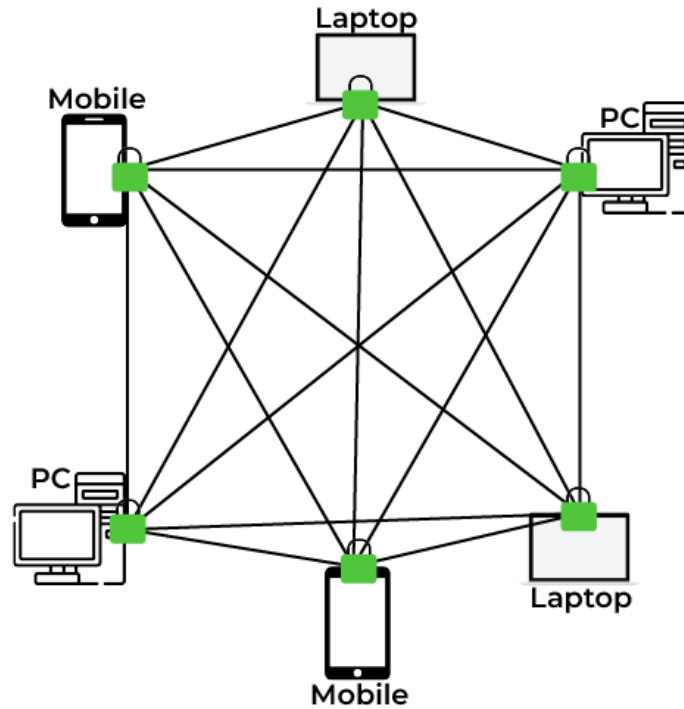


Figure 2.8: Private blockchain [43].

2.2.6.3 Consortium Blockchain:

Unlike private blockchains, which are authorized by a single entity, consortium blockchains are authorized by a number of organizations and the government. In comparison to private blockchains, consortium blockchains are more decentralized, which improves block security and privacy.

Those like private blockchains connected with government organizations block networks. Between public and private blockchains are consortium blockchains. They are created by organizations, and access is restricted to those organizations alone. All businesses in between entities work together equally on consortium blockchains. They don't allow access from outside the organizations / consortium network. [43]

2.2.6.4 Hybrid Blockchain:

Hybrid blockchains combine elements of both public and private blockchains. By permitting transparent and regulated access, they seek to provide the advantages of both kinds. IBM's Food Trust and Dragonchain are two examples. [43]

2.2.7 What are blockchain protocols?

Various blockchain platforms that are available for application development are referred to as blockchain protocols. The fundamental blockchain concepts are modified by each blockchain protocol to fit particular sectors or uses. The following subsections offer a few instances of blockchain protocols: [35]

- **Hyperledger fabric:**Hyperledger Fabric is an open-source project that includes a number of libraries and tools. Businesses can use it to swiftly and efficiently create private blockchain applications. It is a general-purpose, modular framework with special access control and identity management capabilities. Because of these characteristics, it can be used for a wide range of purposes, including supply chain tracking and tracing, trade financing, loyalty and incentives, and financial asset clearing and settlement.
- **Ethereum:**People can create public blockchain applications using Ethereum, a decentralized open-source blockchain platform. Ethereum Enterprise is intended for commercial applications.
- **Corda:**Corda is a business-oriented open-source blockchain initiative. Building interoperable blockchain networks that conduct transactions in complete secrecy is possible with Corda. Companies can conduct valuable, direct transactions by utilizing Corda's smart contract technology. Financial institutions make up the majority of its users.
- **Quorum:** An open-source blockchain protocol called Quorum is based on Ethereum. It is specifically made to be used in either a consortium blockchain network, where several members each own a section of the network, or a private blockchain network, where just one member owns all the nodes.

2.3 Hyperledger fabric

2.3.1 Définition

Developed for usage in enterprise settings, Hyperledger Fabric is an open-source enterprise-grade permissioned distributed ledger technology (DLT) platform that offers several significant advantages over other well-known blockchain or distributed ledger platforms.[\[44\]](#)

2.3.2 Key Features

2.3.2.1 Permissioned Network:

The Fabric platform is permissioned, which means that members are known to one another rather than anonymous and, as a result, completely untrusted, in contrast to a public permissionless network. This implies that a network can function under a governance model that is based on what trust does exist between participants, such as a legal agreement or dispute resolution framework, even though the participants may not fully trust one another (for instance, they may be rivals in the same industry).[\[44\]](#)

2.3.2.2 Modularity:

The architecture of Hyperledger Fabric is deliberately designed to be modular. Whether it be cryptographic libraries, identity management protocols, key management protocols, or consensus, the platform's fundamental design allows it to be set up to satisfy the many needs of enterprise use cases.

The following modular components make up Fabric at a high level:[44]

- After reaching an agreement on the transaction order, a pluggable ordering service publishes blocks to peers.
- The task of assigning cryptographic identities to network entities falls to a pluggable membership service provider.
- By directing service to other peers, an optional peer-to-peer gossip service distributes the block output.
- For isolation, smart contracts, often known as "chaincode," operate in a container environment, such as Docker. They lack direct access to the ledger state, but they can be built in common programming languages.
- A range of DBMSs can be supported by the ledger's configuration.
- An endorsement and validation policy enforcement plug-in that can be set up separately for each application.

The industry is fairly in agreement that there isn't a single blockchain that can govern them all. Hyperledger Fabric can be set up in a variety of ways to meet the various needs of various industry use cases.

2.3.2.3 Channels for Data Privacy:

Because Hyperledger Fabric is a permissioned platform, its private data capability and channel architecture allow for confidentiality. Through channels, users on a Fabric network create a sub-network in which all users may see a certain set of transactions. As a result, the smart contract (chaincode) and the data exchanged are only accessible by the nodes that are part of a channel, protecting their privacy and secrecy. Without the maintenance burden of setting up and maintaining a separate channel, private data enables collections between channel members, providing much of the same privacy as channels.[44]

2.3.2.4 Smart Contracts (Chaincode):

A smart contract, also known as "chaincode" according to Fabric, is a trusted distributed application that derives its security and trust from the blockchain and the underlying peer consensus. It is a blockchain application's business logic.[44]

2.3.2.5 Performance and Scalability:

Transaction execution may be expanded horizontally across peer nodes thanks to Fabric's special execute-order-validate mechanism. Additionally, no expensive mining is necessary because it is a permissioned blockchain. When combined, they indicate that Fabric can function and grow without the limitations of the majority of current distributed ledger solutions. Numerous factors, including transaction, block, and network sizes as well as the hardware resources available (CPU, memory, disk space, disk, and network input/output), can impact a blockchain platform's performance.[44]

2.3.2.6 Pluggable Consensus:

A modular consensus component that is logically separated from the peers who carry out transactions and keep the ledger is tasked with sorting transactions. The ordering service, in particular. Consensus can be implemented in a way that is customized to the trust assumption of a specific deployment or solution since it is modular. The platform can rely on established toolkits for either BFT (byzantine fault-tolerant) or CFT (crash fault-tolerant) ordering because to its modular architecture.[44]

2.3.2.7 Identity Management via MSP:

Blockchain users must have a means of proving their identity to the rest of the network in order to conduct transactions on Fabric since it is a permissioned network. By creating a public and private key combination that may be used to verify identification, certificate authorities are able to issue IDs. The MSP is necessary in order for the network to recognize this identity. For instance, a peer digitally signs, or endorses, a transaction using its private key. The peer's authorization to approve the transaction is verified by the MSP. The validity of the signature affixed to the transaction is then confirmed using the public key from the peer's certificate. Therefore, the MSP is the mechanism that enables the rest of the network to trust and recognize that identity.[45]

So the MSP manage the identities and roles within the network.

2.3.2.8 Rich Query Capabilities:

CouchDB may be used as the state database in Hyperledger Fabric, enabling you to describe data on the ledger as JSON and run rich queries against data values instead of keys.[46]

2.3.2.9 Endorsement policies:

The set of peers on a channel that must execute a chaincode and approve the execution outcomes for the transaction to be deemed legitimate is specified by the endorsement policy for each chaincode.[47]

2.3.3 Key Components of Hyperledger Fabric

2.3.3.1 Organizations:

An organization is a group of people who are logically governed. This could be as tiny as a flower shop or as large as a multinational enterprise. The fact that organizations (or orgs) manage their members under a single MSP is what matters most. An identity can be connected to an organization through the MSP.[\[45\]](#)

So in Hlf Network,multiple organizations collaborate to maintain the blockchain.each organization:

- Own and manages its own peers(nodes).
- Defines access policies and roles.
- Participates in consensus and transaction validation.

2.3.3.2 Peers(Nodes):

- **Endorsing peers:** Validate and simulate transactions before they are committed.
- **Committing peers:** Store the blockchain ledger and commit transactions.
- **Anchor peers:** Facilitate communication between organizations within a channel.

2.3.3.3 Orderer Nodes(ordering Service):

The orderer service is a important component of hyperledger fabric as its ensures that transactions are correctly ordered and sent to all peers in the network .

→ 2.2.3.3.1 What does the ordering service do?:

- **Receiving transactions:** Collecting transactions from different organizations.
- **Transactions request:** Determine the sequence of transactions before adding them to the ledger.
- **Handing over transactions to peers:** Collecting transactions in block and send it to all peers to validate it and commit it.

→ 2.2.3.3.2 The consensus mechanisms in ordering service:

The ordering service use the different consensus mechanisms to achieve agreement between participants in network:

- **Raft (most common):** Leader-driven mechanism for high availability and fault tolerance.
- **Kafka:** Uses message list to request transactions.
- **Bft:** Protect against malicious parties, but is more complex.

2.3.3.4 Channels:

Channels are private, segregated communication conduits in Hyperledger Fabric that allow particular users to share data and transactions in a permissioned blockchain network. They are essential for maintaining data segregation, scalability, and privacy in distributed ledger environments.

Participants in Hyperledger Fabric may be part of one or more channels, each of which has its own configuration, smart contracts (chaincode), and ledger. Only those participants who have been given access to a particular channel can see transactions carried out within it, guaranteeing private communications between pertinent parties.[\[48\]](#)

2.3.3.5 Chaincode(Smart contract):

A program that implements a specified interface that is written in Go, Node.js, or Java is called chaincode. A secure Docker container apart from the endorsing peer process is where Chaincode operates. Application-submitted transactions are used by Chaincode to initialize and maintain the ledger state. A chaincode can be thought of as a "smart contract" since it usually manages business logic that has been agreed upon by network participants. A chaincode's produced state is scoped just to that chaincode and is not directly accessible by another chaincode. However, a chaincode may call another chaincode to access its state inside the same network if it has the necessary authorization.[\[49\]](#)

2.3.3.6 MSP:

MSPs, or membership service providers, are essential to the network's identity and access control management. In Hyperledger Fabric, membership services are in charge of controlling participant identities, making sure that only legitimate and trustworthy organizations are able to use the network. Membership services are crucial for preserving the blockchain's integrity and shielding it from bad actors. Permissioned blockchains need a strict authentication process to limit who can transact and access data on the network, in contrast to public blockchains, where anybody can sign up and take part.[\[50\]](#)

2.3.4 How organizations collaborate in a fabric network ?

2.3.4.1 Establish policies and MSP:

As the network's gatekeepers, MSPs verify each participant's legitimacy and authenticity. They specify the guidelines and procedures that control the management, verification, and revocation of identities when needed. Every network participant organization has its own MSP, which establishes the requirements for membership and decides each participant's degree of access.[\[50\]](#)

2.3.4.2 Network creation and configuration:

Organizations agree on the network structure, creating channels ,deploy the ordering service.

- **Consortium Formation:** A group of organizations form a consortium to create the network, and when the network is first set up, the consortium agrees on a set of policies that govern each organization's permissions. Furthermore, as we'll learn when we talk about the idea of modification policy, network policies are subject to change over time with the consent of the consortium's organizations. [51]
- **Channel Creation:** An enterprise must join a channel in order to produce and transfer assets on a Hyperledger Fabric network. Other network users cannot see channels, which are a private layer of communication between particular businesses. Every channel has its own ledger that only channel members can read and write to. They may also join other channel members and get fresh transaction blocks from the ordering service. Channels are how companies connect and communicate with one another, whereas peers, nodes, and Certificate Authorities make up the network's physical architecture. [52]
- **Ordering Service Deployment:** Since many distributed blockchains, like Ethereum and Bitcoin, lack permissions, any node can take part in the consensus process, which arranges transactions into blocks. These systems rely on probabilistic consensus algorithms, which eventually ensure ledger consistency to a high degree of probability. However, these algorithms are still susceptible to divergent ledgers, also referred to as a ledger "fork," in which various network participants hold differing opinions about the accepted order of transactions.

Hyperledger Fabric functions in a unique way. It has an orderer node (sometimes referred to as a "ordering node") that does this transaction ordering; when combined with additional orderer nodes, it becomes an ordering service. Any block verified by the peer is certain to be final and accurate since Fabric's design is based on deterministic consensus techniques. Unlike many other distributed and permissionless blockchain networks, ledgers are unable to fork. [53]

2.3.4.3 Deployment of smart contract(chaincode):

The code that apps use to read and change data on the blockchain ledger is called a smart contract . It is packaged as chaincode. Business logic is transformed into an executable program via a smart contract, which is accepted and validated by all peers or members of a blockchain network.[54]

2.3.4.4 Transaction flow:

A client-side application initiates the data flow for queries and transactions on a Hyperledger Fabric network by sending a transaction request to a peer on a channel.

Both inquiries and transactions share the same beginning data flow across the network:[55]

- A client application signs and sends a transaction proposal to the relevant endorsing peers on the designated channel using APIs found in the SDK. This first proposal for a transaction is an endorsement request.

- After confirming the identity and legitimacy of the submitting client, each peer on the channel compares the provided inputs to the defined chaincode. Each peer responds to the application with a signed YES or NO depending on the endorsement policy for the invoked chaincode and the transaction results. Every YES response that is signed signifies that the transaction is approved.
- The procedure for inquiries and transactions changes at this stage of the transaction flow. The application gives the client the data if the proposal invoked a query function in the chaincode. The following actions are taken by the application if the proposal updated the ledger by calling a function in the chaincode:
- The transaction, along with the read/write set and endorsements, is sent to the ordering service by the application. After that, the ordering service receives the transaction. Every channel peer performs a Concurrency Control Version Check and applies the chaincode-specific Validation Policy to verify every transaction in the block.
- The block is appended to the channel's ledger and flagged as invalid for any transaction that fails the validation process. Every legitimate transaction uses the updated key/value pairs to update the state database appropriately.

2.3.5 Benefits of Hyperledger fabric

- **Permissioned network :**
Instead of using an open network with anonymous users, create decentralized trust in a network with known participants. [56]
- **Confidential transactions :**
Give only the information you desire to the people you wish to share it with. [56]
- **Pluggable architecture :**
Instead of using a one-size-fits-all strategy, use a pluggable architecture to adapt the blockchain to industry demands. [56]
- **Simple to get started:**
Rather than learning new languages and architectures, create smart contracts in the languages your team now uses. [56]

2.3.6 Use Cases

- **Supply Chain Management:**
permits traceability and transparency across the supply chain, making it possible to follow products from point of origin to point of consumption. It boosts productivity, lowers fraud, and enhances product quality. [57]
- **Identity Management:**
improves security and lowers operating costs by enabling decentralized identity systems without centralized authorities. [57]

- **Healthcare:**
permits providers to share patient data in a safe and secure manner. This lowers expenses, gets rid of duplications, and enhances healthcare service. [57]
- **Banking and Finance:**
facilitates safe and open financial transactions. It minimizes transaction costs, decreases fraud, and improves payment efficiency. [57]
- **Insurance:**
enhances customer satisfaction and expedites claim processing by preventing fraud and enhancing claims administration through a decentralized system. [57]
- **Real Estate:**
reduces fraud, increases transparency, and streamlines the real estate process by offering a safe platform for real estate transactions. [57]
- **Government:**
can be applied to reduce corruption, track public monies transparently, and provide safe voting systems. [57]
- **Energy:**
improves billing accuracy, optimizes distribution, and permits decentralized tracking of energy output and consumption. [57]
- **Retail:**
increases supply chain visibility, facilitates safe loyalty programs, and improves inventory and transaction tracking. [57]
- **Intellectual Property:**
uses a decentralized ledger to assist in managing patents and copyrights, lowering the risk of infringement and expediting intellectual property transactions. [57]

2.4 State of the art

2.4.1 Bindra et al. (2020) Flexible, Decentralized Access Control for Smart Buildings with Smart Contracts

Bindra et al. (2020) Blockchain-Based Access Control Using Ethereum This academic solution proposes a decentralized access control system built on the Ethereum blockchain. In it, smart contracts are used to manage who can enter smart buildings. Tenants (users) can grant or revoke access to other individuals without needing a centralized administrator. Every access event is recorded as a transaction on the blockchain, ensuring transparency and immutability. It supports access delegation (e.g., a tenant granting access to a guest) and revocation. Since it is built on Ethereum, users must pay gas fees, and the system requires internet connectivity.[58]

2.4.2 NineSmart Smart Access | QR Code Access Control

NineSmart QR CodeBased Smart Access (Commercial Startup) NineSmart offers a commercial access control solution using QR codes. It is centralized and cloud-based, meaning a building manager or tenant can generate and share QR codes via a mobile app. These QR codes are scanned at doors using smart readers. The entire system runs through a cloud dashboard, making it easy to manage access remotely. However, this system does not use blockchain, so it lacks decentralization, transparency, or tamper-proof logging.[59]

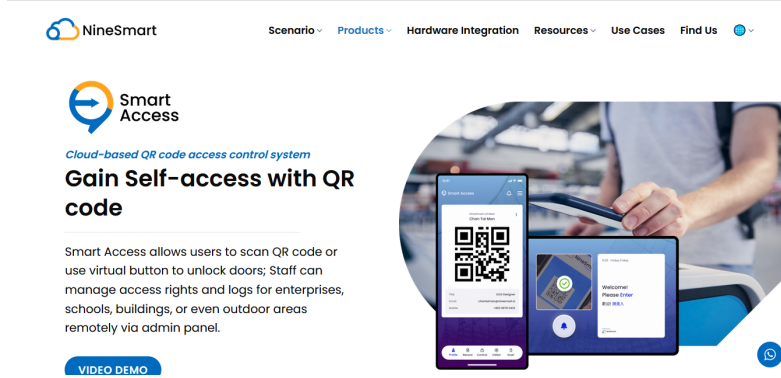


Figure 2.9: NineSmart

2.4.3 SPCL: A Smart Access Control System That Supports Blockchain

The SPCL system uses a private blockchain to manage access to smart locks. Every entry and access rule is stored on-chain. The system does not use QR codes or NFC; instead, it relies on blockchain identity and time-stamped records to determine who can unlock a door and when. It supports smart lock integration directly and ensures access traceability. However, the design is limited to simple access rules and lacks features like visitor management or dynamic QR authentication.[60]

2.5 Comparison table

Table 2.1: Comparative Summary of Access Control Solutions

Feature	Bindra et al. (2020)	NineSmart (Startup)	SPCL System	Our Project (HLF)
Blockchain Type	Public Ethereum	No Blockchain	Private Blockchain	Private (Hyperledger Fabric)
Smart Contract Use	✓ Yes (Ethereum)	×	✓ Yes	✓ Yes (Chaincode)
Access Methods	Mobile App / Token	QR Code	Blockchain-based Locks	NFC Card + QR Code
User Types	Tenants, Visitors	Visitors	Not specified	Residents, Visitors, Requested Visitors
Access Approval Work-flow	Delegated via contracts	Managed via cloud	Predefined policy rules	Manager / Resident Approval
QR Code Support	✓ Yes	✓ Yes	×	✓ Yes
NFC Card Support	×	×	×	✓ Yes
Remote Access	×	✓ Yes	×	×
Internet Required	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Transaction Fees	High (Gas fees)	None	None	None

Our project offers a robust, secure, and flexible access control solution specifically designed for smart buildings in Algeria. It bridges the gap between academic proposals (like Bindra et al.) and commercial solutions (like NineSmart) by combining blockchain security with modern authentication methods (NFC and QR), while maintaining privacy and role-based governance.

2.6 Conclusion

In conclusion, Hyperledger Fabric persists as a prominent manifestation of blockchain technology, distinguished by its modular architecture, emphasis on security, transparency, and support for privacy-preserving transactions. These characteristics make it especially well-suited for implementing robust access control systems in smart buildings .

The amalgamation of decentralization, identity management, and immutable record-keeping renders Hyperledger Fabric highly desirable for institutions aiming to enhance trust and efficiency in digital access environments.

In the next chapter, we will present the conception and development of our blockchain-based access control system using Hyperledger Fabric, which constitutes the core objective of our project.

Chapter 3

Proposed solution

3.1 Introduction

In the previous chapter, we introduced blockchain technology and the Hyperledger Fabric platform, that we have selected for implementing our solution.

In this chapter, we move on to a more detailed presentation of our proposed system. The system is designed to manage access control within a building by identifying only authorized individuals who are allowed to enter, thus ensuring a higher level of security and traceability.

First, we will introduce the global architecture of our system, describing its main components and how they interact. Following that, we will provide a detailed overview of the Fabric network, including the participating organizations, peers, ordering service, and other network elements, and then we will explain the role of the chaincode. Additionally, we will include diagrams to provide a clear understanding of the system.

3.2 Use Case diagram

The use case diagram presented below (figure 3.) provides an overview of the main functionalities of the access control system by illustrating the interactions between the system and its users. It defines the roles of each actor and the services they can access, helping to clarify how the system operates. This diagram serves as a foundation for understanding the different actions performed by the Admin, Resident, Visitor, and Requested Visitor within the system.

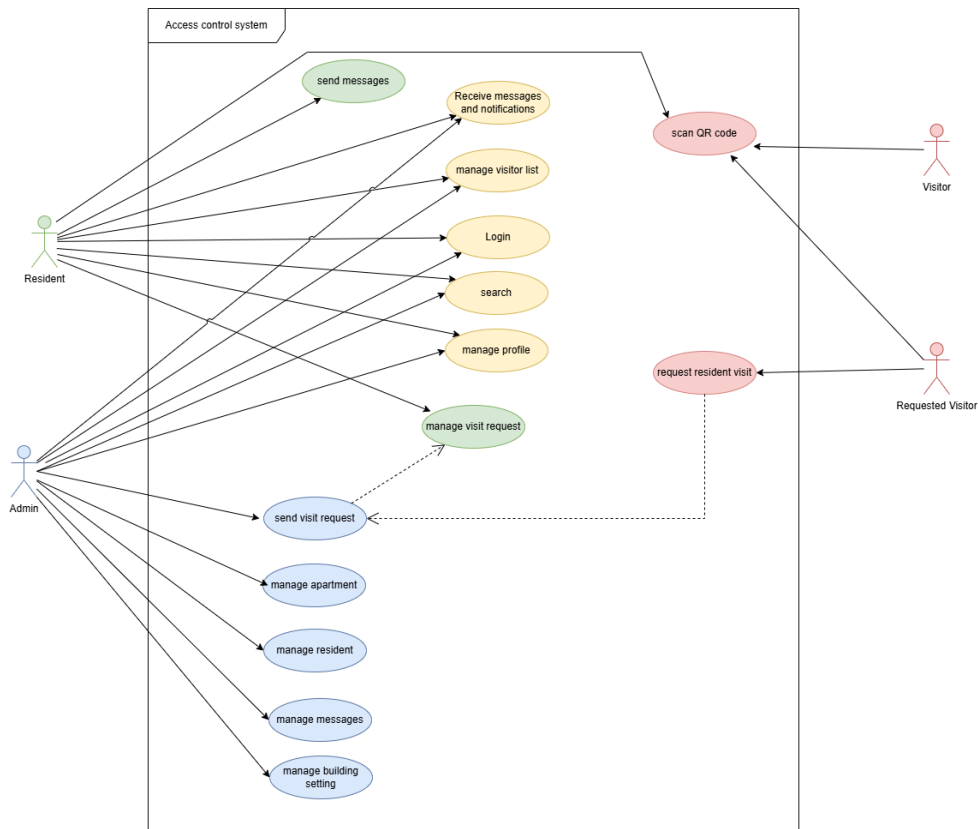


Figure 3.1: Use Case diagram

3.2.0.1 Admin:

- **Manage Building Settings:** By default, the system is initialized with configuration parameters such as the building's name, number of apartments, residents, and the maximum number of visitors allowed per resident. The admin can modify these settings.
- **Manage Apartment:** Add, delete, or edit apartment.
- **Manage Resident:** Admin can:
 - Add new residents (a permanent QR code is generated).
 - Edit, delete, block/unblock residents.
- **Manage Messages:** View and respond to messages received from residents.
- **Send Visit Request:** When a Requested Visitor (not listed in any residents visitor list) arrives at the building, the Admin sends a visit request to the corresponding resident for approval.

3.2.0.2 Resident:

- **Send Messages:** to communicate with the admin.

- Manage Visit Requests: Accept or reject visit requests. If accepted, a temporary QR code is generated for the visitor.

3.2.0.3 Common Functionalities:

- Shared by Admin and Resident:

- Login: Both must log in to access and perform tasks.
- Manage Visitor List:
 - Add visitors (temporary QR codes generated).
 - Edit, delete, block/unblock visitors.
- Receive Messages and Notifications: Both are notified of events/messages.
- Search Query system data.
- Manage Profile: Modify personal information and password.

- Shared by Resident, Visitor and Requested Visitor:

- Scan QR Code: All scan codes to access the building.

3.2.0.4 Requested Visitor:

- Request Resident Visit: Requests access via the admin, who relays the request to the resident.

3.3 Global architecture of the system

The proposed system follows a layered architecture composed of four main components, as illustrated in Figure 3.2. At the user level, there are two primary actors: the Manager and the Resident. Both interact with the system through a graphical interface, which facilitates all user operations. The interface sends requests to a server, which acts as a middleware, handling logic and forwarding valid requests to the blockchain network.

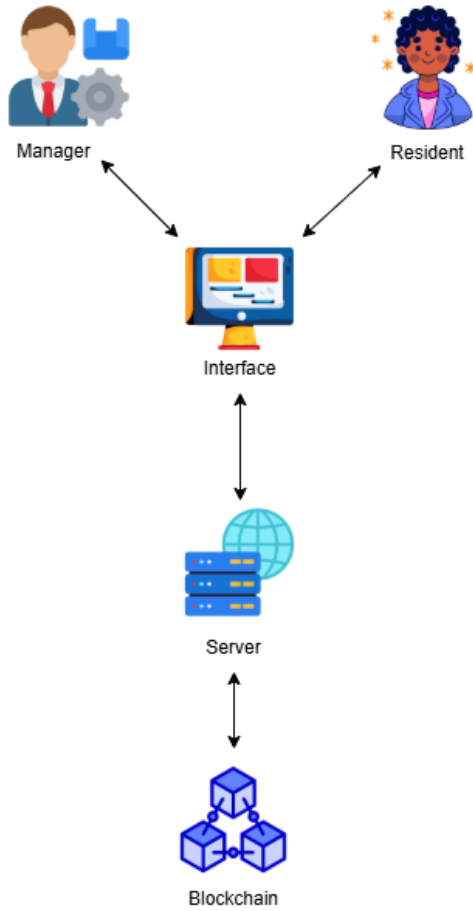


Figure 3.2: Global Architecture

3.4 Detailed architecture of the system

The system is built on Hyperledger Fabric and consists of two organizations: Residents, Manager. Each organization plays a distinct role within the blockchain network, collaborating through defined channels and smart contracts to ensure secure, transparent, and decentralized management of access control in the smart building environment.

3.4.1 Hyperledger Fabric Network

3.4.1.1 Organizations:

- **Residents:** Responsible for representing the residents who manage visitor access to their homes. Each resident has a digital identity that allows them to approve or reject entry requests. The organization stores resident data and their visitor lists on the distributed ledger and includes peers to process transactions.
- **Manager:** Responsible for overseeing access control for both visitors and residents. It has the authority to approve or deny new visitor entry requests and contributes to defining and enforcing custom access rules for enhanced security.

3.4.1.2 Ordering Service:

Responsible for ordering transactions and coordinating their addition to the distributed ledger.

In this system, the Raft consensus mechanism is used to ensure fault tolerance and prevent transaction loss.

so orderer Handles all transaction requests from peers and forwards valid transactions to be committed to the ledger.

3.4.1.3 Peers:

Each organization in the network contains **Peers**, which are responsible for:

- Hosting the distributed ledger (Ledger).
- Executing smart contracts (Chaincode).
- Validating transactions and operations.

Peer Distribution in the Network:

Organization	Peer	Responsibility
Org1 (Residents)	peer0.residents.example.com	Handles resident data and visitor lists.
	peer1.residents.example.com	Acts as a backup peer to ensure network stability.
Org2 (Building Manager)	peer0.manager.example.com	Manages access permissions and approves visitor entries.

Table 3.1: Distribution of Peers and their Responsibilities

3.4.1.4 Channels:

are used to ensure the confidentiality and security of transactions between different organizations. In this system, we have :

- **residentschannel:** Includes **Org1 (Residents)** and **Org2 (Manager)**. It is used to manage visitor lists and approve access requests.

3.4.1.5 MSP(Membership Service Provider):

is the system responsible for managing the digital identity of every user or node in the network.

MSP Distribution in the Network:

MSP Name	Description
ResidentsMSP	For residents who manage their visitor lists and access rights.
ManagerMSP	For the building manager who approves or denies access requests.
OrdererMSP	For managing the transaction ordering process in the network.

Table 3.2: MSP Assignment and Roles

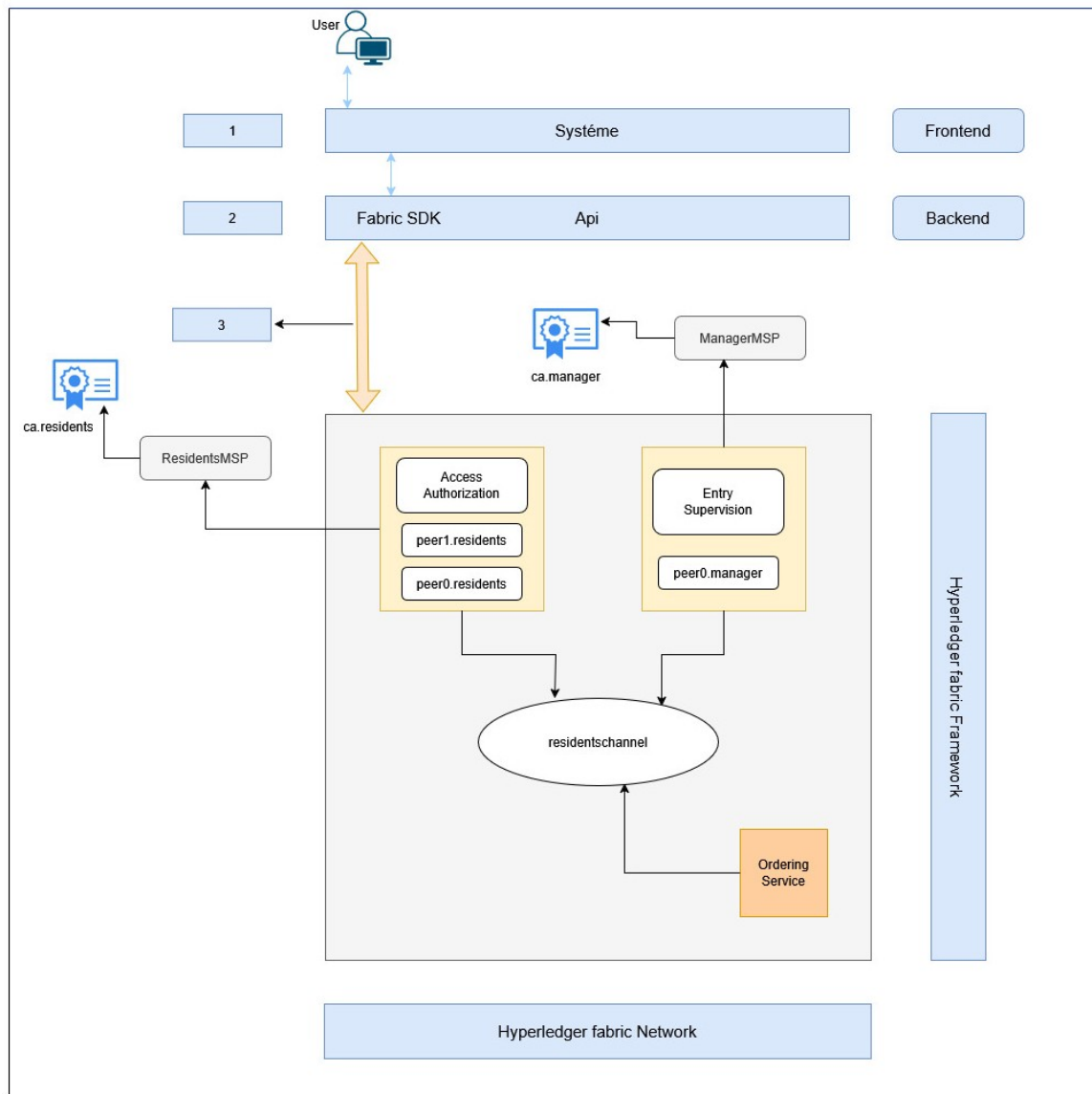


Figure 3.3: Deatiled architecture of the system

3.5 Hyperledger Fabric Network Setup and Development Steps

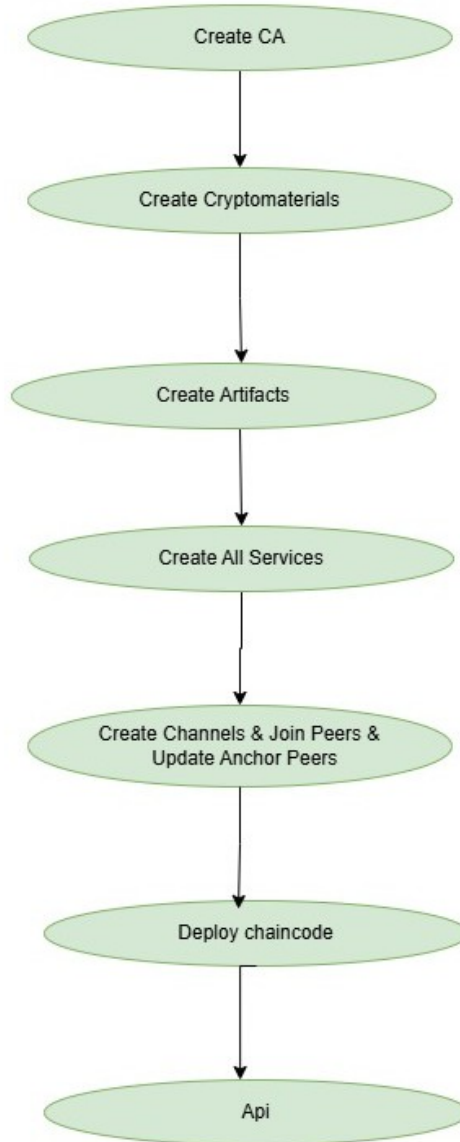


Figure 3.4: Development Steps of Hlf

- **Create CA:** The Certificate Authority, which provides the required cryptographic identities for users participating in the HLF network, is set up in this stage. The Fabric CA Server is used.
- **Create Cryptomaterials:** Each participant receives cryptographic materials, including keys and certificates, upon the formation of the CA. Making use of the cryptogen tool, a straightforward tool for creating crypto materials.
- **Create Artifacts:** Artifacts like the channel configuration, genesis block, and other required files are made in order to bootstrap the network.

- **Create All Services:** This entails starting up every network component, including orderers, peers, and other services.
- **Create Channels & Join Peers & Update Anchor Peers:** Create channels, join peers to channels, and update anchor peers to establish a private, structured communication network where specific peers can securely interact and discover each other within the Hyperledger Fabric blockchain.
- **Deploy chaincode(Smart contract):** The smart contract, also known as chaincode, is implemented on the channel to outline the logic for managing transactions.
- **Api:** Api compose 3 steps like RegisterUser, Invoke Transaction, Query. So RegisterUser sets up the user's identity, Invoke Transaction updates the blockchain, and Query retrieves data for verification forming the standard client interaction pattern with Fabric.

3.6 Deploy Chaincode(Smart Contract)

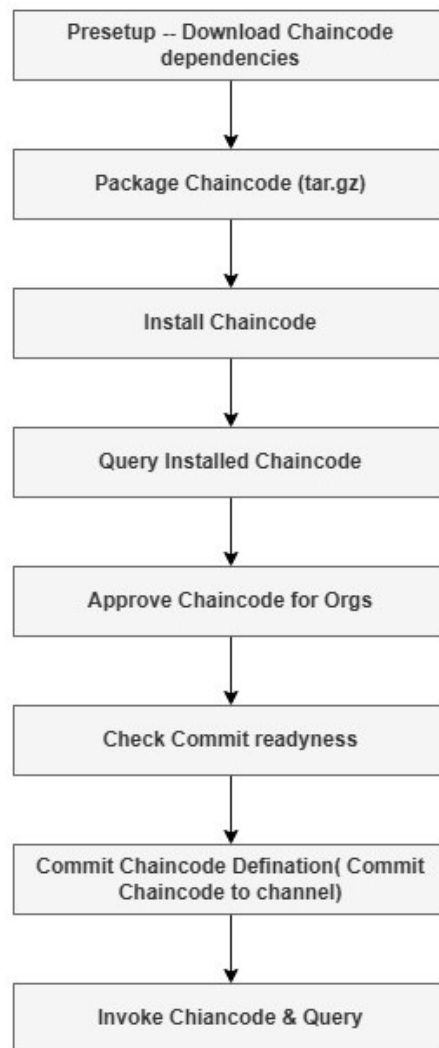


Figure 3.5: Deploy Chaincode Steps

- Chaincode Dependency Setup (Presetup): Before deploying the chaincode, dependencies such as Go modules or Node.js packages must be downloaded and prepared. This ensures the chaincode has everything needed to compile and run correctly on peers.
- Package the Chaincode (Package Chaincode): The developed chaincode is packaged into a `.tar.gz` archive. This package includes the source code and metadata required for deployment. It serves as the artifact to be installed on peers.
- Install Chaincode on Peers: The packaged chaincode is installed on the peer nodes of each participating organization in the network. This installation allows each peer to recognize and prepare to execute the chaincode.
- Query Installed Chaincode: After installation, peers are queried to confirm that the chaincode has been installed successfully. This step helps retrieve the package ID, which is required for chaincode approval.
- Approve Chaincode Definition for Organization: Each organization must formally approve the chaincode definition. This includes specifying the name, version, sequence number, and endorsement policy. Approval ensures agreement on how the chaincode behaves and who must endorse transactions.
- Check Commit Readiness: Before committing the chaincode, the network checks whether all necessary organizations have approved the chaincode definition. This step ensures consensus and readiness to proceed.
- Commit Chaincode to Channel: Once all approvals are received, the chaincode definition is committed to the channel. This makes the chaincode operational across the channel and available to all peers for execution.
- Invoke Chaincode and Query (Smart Contract Execution): After deployment, the chaincode can be invoked to perform write operations (transactions) or queried for read-only access to ledger data. These operations validate the smart contract logic and confirm proper integration.

3.7 Application Programming Interface(Api)

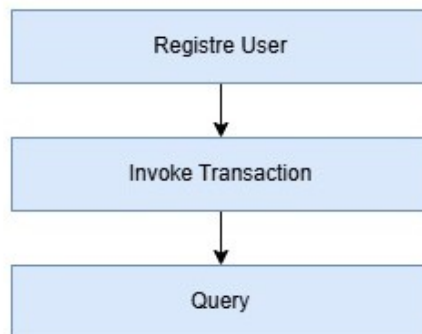


Figure 3.6: Api

- Register User: This is the first step where a new user identity is registered with the Fabric Certificate Authority (CA). It involves:
 - Enrolling the admin identity (if not already done).
 - Registering a new user with the CA using admin credentials.
 - Enrolling the user to obtain a certificate and private key.
 - Storing the user credentials in a wallet for later use.
- Invoke Transaction: Once the user is registered and enrolled, they can submit transactions to the blockchain. This step:
 - Loads the user identity from the wallet.
 - Submits a transaction proposal to peers for endorsement.
 - Sends the endorsed transaction to the orderer to be committed to the ledger.
- Query: This step is used to fetch data from the ledger without making any changes. It involves:
 - Using the registered user identity to send a query request.
 - Getting a response directly from the peer based on current world state data.
 - Verifying data such as transaction results, asset ownership, or access permissions.

3.8 Functionality of the system

To better understand the functionality of the system, it's best to represent it in diagrammatic form. In this section, we'll present a representation of our system in the form of a use case, a sequence diagram (Figure 3). This diagram shows all the actions that an administrator, a resident, a visitor, and the visitor sending the request can perform.

3.8.1 Sequence diagram of the system

The following sequence diagram illustrates the interaction between the main actors of the system: Admin, Resident, Visitor, and Requested Visitor. It is structured into five steps, starting from the initial system setup through to daily operations.

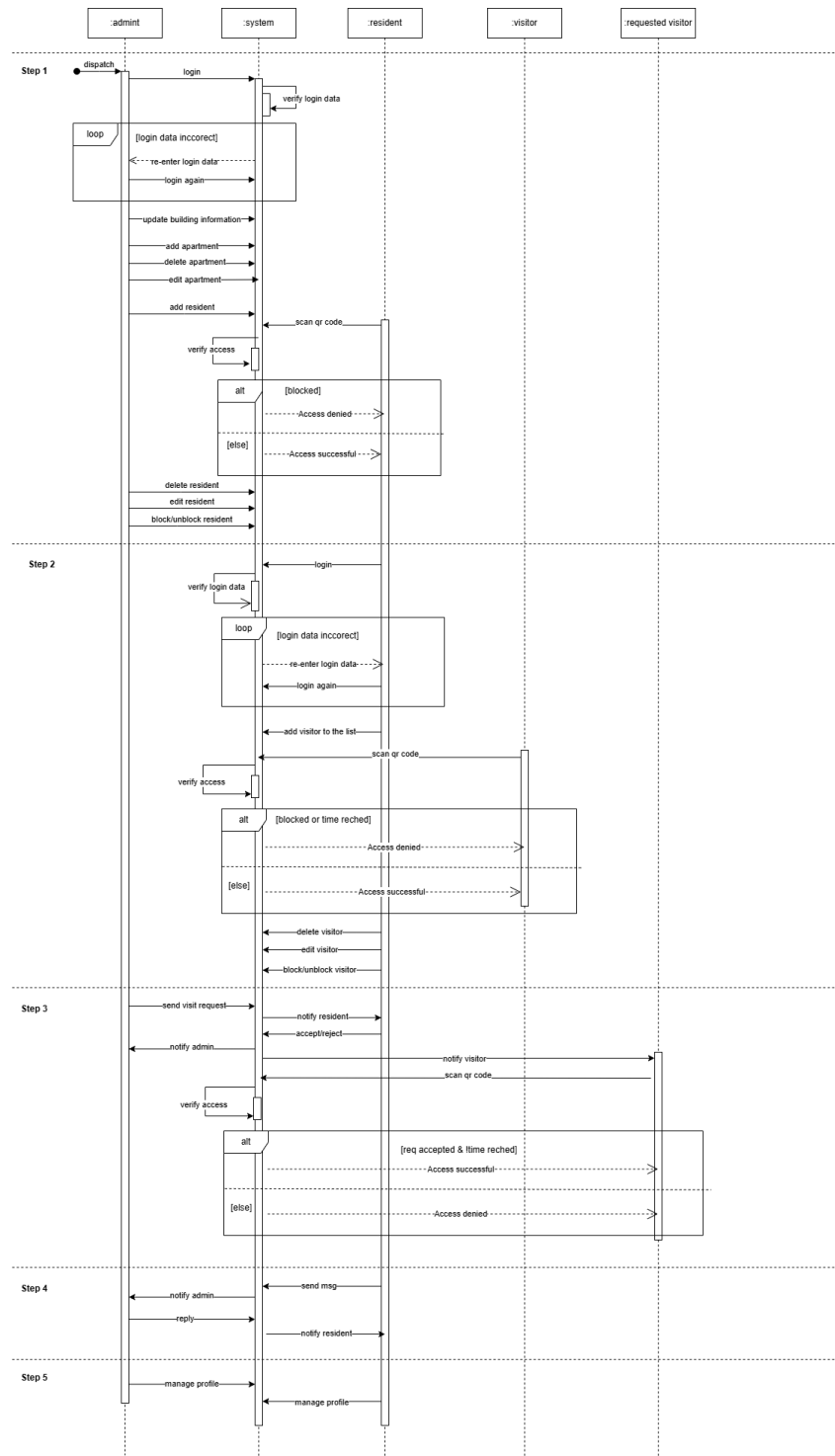


Figure 3.7: Sequence diagram

3.8.1.1 Step 1:

- The Admin begins by logging into their account. If the login credentials are incorrect, the system prompts for a retry until successful authentication is achieved.
- Upon successful login, the system loads with default configuration settings (building name, apartment count, resident count, visitor count). The Admin may use or

modify these settings.

- The Admin then proceeds to:
 - Add new apartments.
 - He can also edit or delete apartments.
 - Add residents to specific apartments.
 - When a resident is added, the system generates a permanent QR code.
 - This QR code is intended to be sent via SMS for security purposes. However, due to limitations with SMS services in Algeria and local testing conditions (localhost environment), this feature is simulated rather than operational during development.
- Once the resident receives the QR code, they can scan it at the building entrance. The system verifies the access:
 - If access is blocked, entry is denied.
 - Otherwise, access is granted.
- The Admin can also manage residents by editing, deleting, or blocking/unblocking them.

3.8.1.2 Step 2:

- The Resident logs in to their account. As with the admin, login attempts are verified until successful.
- Once logged in, the resident can begin managing their visitor list:
 - Adding a visitor generates a temporary QR code tied to a specific visit time.
 - This QR code is to be sent to the visitor via SMS (simulated for now).
- The visitor uses the QR code to attempt access to the building. The system checks:
 - Whether the QR code is blocked or invalid (visit time has passed).
 - If valid, the visitor is granted access; otherwise, access is denied.
- Residents can also edit, delete, or block/unblock visitors from their list.

3.8.1.3 Step3 :

- A Requested Visitor (someone not already on a Residents visitor list) arrives at the building.
- They request access from the Admin, who sends a visit request to the relevant Resident.
- The system notifies the Resident of the request. The Resident can then accept or reject it.

- The Residents decision is sent back to the Admin. If the request is accepted:
 - A temporary QR code is generated and shared with the requested visitor.
 - This QR code includes a time limit, meaning it can expire if not used within the allowed time.
- When the requested visitor scans the QR code:
 - The system checks whether the request was accepted and whether the QR code is still within its valid time.
 - If both checks pass, access is granted. Otherwise, access is denied (if the code has expired or was rejected).

3.8.1.4 Step 4:

- The Resident sends messages to the Admin.
- The system notifies the Admin, who can reply.
- The Resident is then notified of the response.

3.8.1.5 Step 5:

- Both Admin and Resident have the ability to manage their user profiles.
- This includes updating account details such as passwords for better security.

3.8.2 Sequence diagram (Hlf)

3.8.2.1 Add Residents:

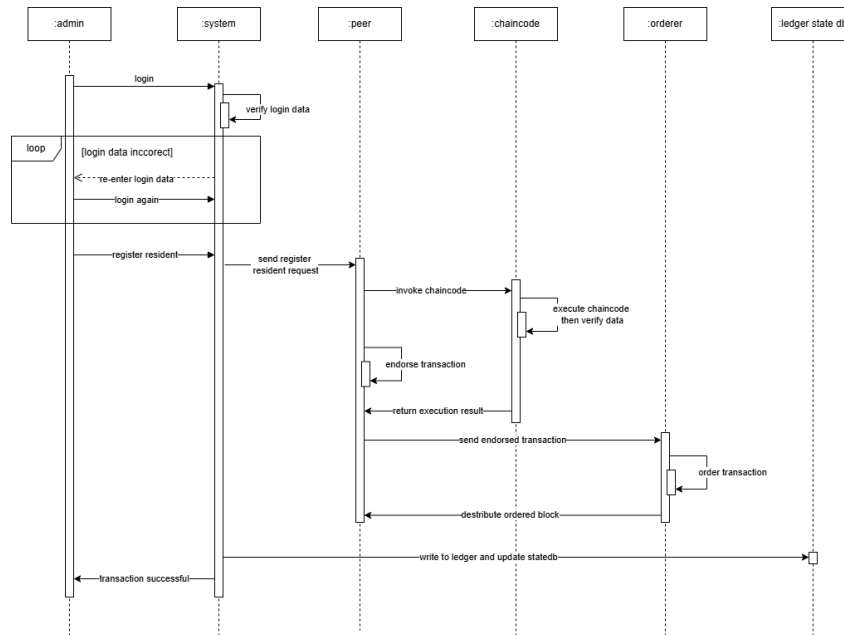


Figure 3.8: Add Residents to stock in ledger

- **Admin/System Initiation:**

- The Admin successfully connects to the application interface.
- The Admin fills out the resident details (e.g., name, apartment, phone) and submits a "Register Resident" request to the system.

- **System to Peer:**

- The application sends the registration request to the connected peer node.
- The peer invokes the RegisterResident chaincode function with the provided resident data.

- **Chaincode Execution:**

- The chaincode verifies if the resident already exists using GetState.
- If not found, it creates a Resident object, generates a QR code, and prepares it for storage.
- The peer endorses the transaction and returns the result.

- **Ordering Phase:**

- The endorsed transaction is sent to the Orderer.
- The Orderer validates, sequences, and packages the transaction into a block.

- **Block Distribution:**

- The new block is distributed to all peers.
- Each peer writes the block to the ledger and updates the world state database with the new resident information.
- **Confirmation:**
 - The System confirms that the transaction was successful.

3.8.2.2 Update Resident:

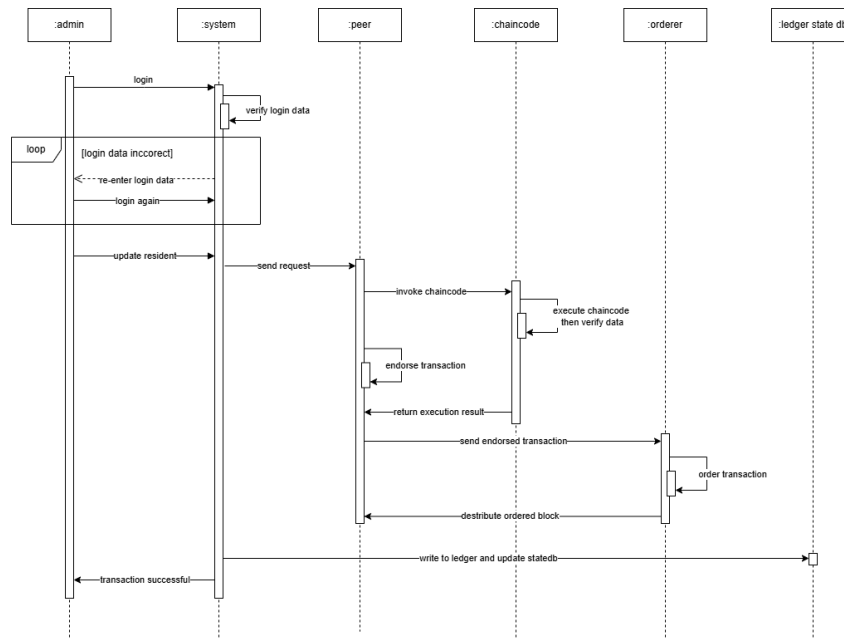


Figure 3.9: Update Resident Informations

- **Admin/System Initiation:**
 - The Admin connects to the system and selects a registered resident to update.
 - The Admin edits fields like email, phone, apartment, or marital status ...etc, then submits the "Update Resident" request.
- **System to Peer:**
 - The system sends the update request to the peer node, specifying the residentId and new data.
 - The peer invokes the UpdateResident function in the chaincode.
- **Chaincode Execution:**
 - The chaincode retrieves the existing resident using `GetState("RESIDENT_" + residentId)`.
 - If the resident exists, it checks if the apartment has changed.

- If so, it fetches the building configuration to ensure the new apartment is not over capacity.
- The residents information is updated, and the `UpdatedAt` timestamp is refreshed.
- **Transaction Endorsement:**
 - The peer endorses the transaction and sends it back to the System.
- **Ordering and Block Formation:**
 - The transaction is submitted to the Orderer, which sequences and batches it into a block.
- **Block Distribution and Ledger Update:**
 - Peers receive the ordered block.
 - Each peer writes the updated resident data into the ledger and updates the state database.
- **Confirmation to Admin:**
 - The system confirms that the update was successful.

3.8.2.3 Block/Unblock Resident:

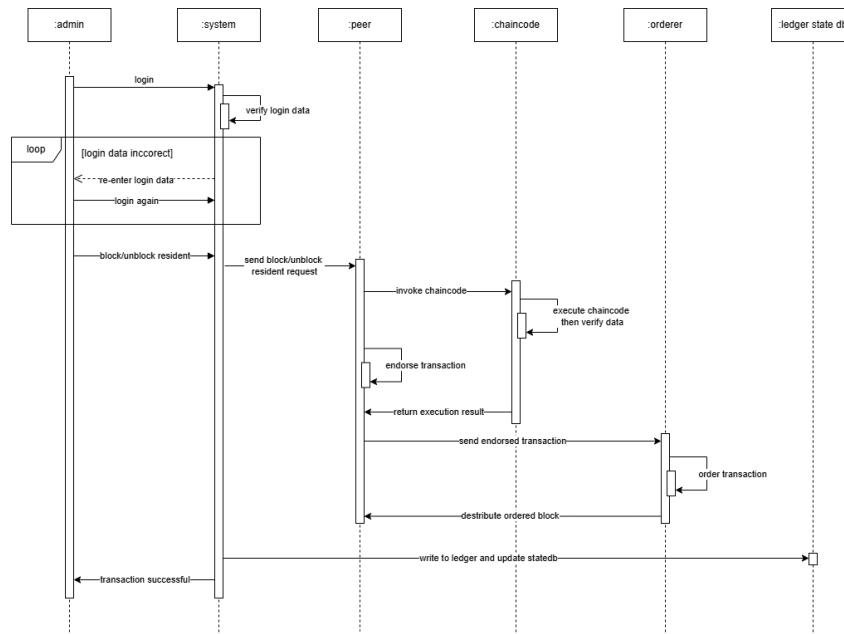


Figure 3.10: Block/Unblock Resident

- **Block Resident:**
 - The admin selects a resident and initiates a block request with inputs: `residentId`, `reason`, `blockedBy`, `fromDateTime`, and `toDateTime`.

- The system sends the block request to the peer node which invokes the **BlockResident** chaincode function.
- The chaincode verifies if the resident exists and if a block already exists. If not, it creates a **BlockRecord** and updates the resident's **isBlocked** flag to **true**.
- The transaction is endorsed, ordered by the Orderer, and committed to the ledger.
- The block record is stored and the ledger state is updated.
- The admin receives confirmation: *Resident successfully blocked.*

- **Unlock Resident:**

- The admin initiates an unblock request by providing the **residentId**.
- The system sends the request to the peer which invokes the **UnblockResident** chaincode function.
- The chaincode checks if the resident is currently blocked, then removes the block record and sets **isBlocked** to **false**.
- The transaction is endorsed, ordered, and added to the ledger.
- The resident's record is updated and the block is removed from the ledger.
- The admin receives confirmation: *Resident successfully unblocked.*

3.8.2.4 Add visitor in each resident:

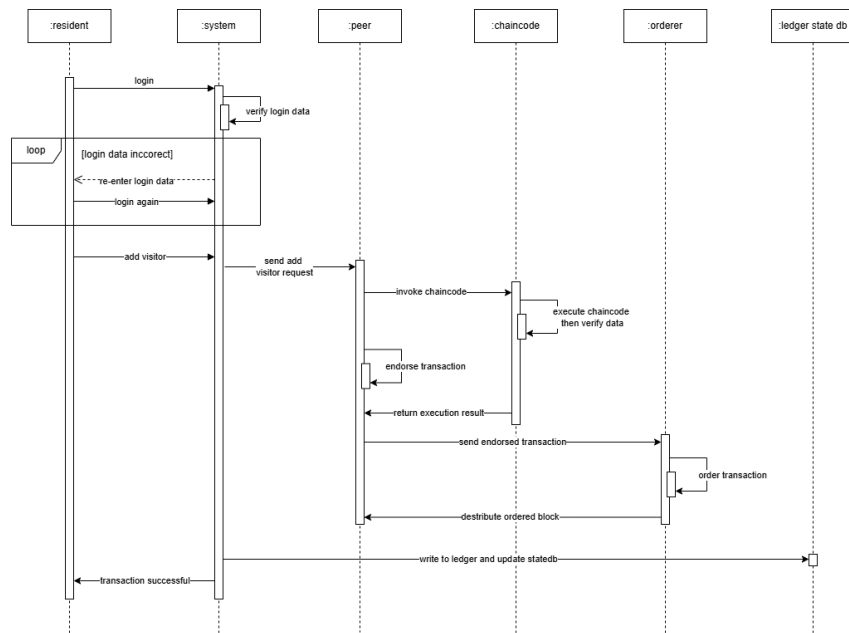


Figure 3.11: Add Visitor

- **Resident/System Initiation:**

- The resident logs into the web application and navigates to the Add Visitor section.
- The resident fills out visitor details: `visitorId`, `fullName`, `phone`, `visitTimeFrom`, `visitTimeTo`, and `relationship`.
- The resident submits an Add Visitor request for a specific `residentId`.
- **System to Peer:**
 - The application sends the visitor registration request to the connected peer node.
 - The peer invokes the `AddVisitor` chaincode function, passing the resident and visitor information.
- **Chaincode Execution:**
 - The chaincode retrieves the resident using `GetState("RESIDENT_" + residentId)`.
 - It checks if the resident exists and whether the visitor already exists in the residents visitor list.
 - If valid, it creates a new `VisitorInfo` object, assigns a QR code (based on `visitorId`), sets the status to Active, and appends the visitor to the residents list.
 - The `UpdatedAt` timestamp for the resident is refreshed.
 - The peer endorses the transaction and returns the result.
- **Ordering Phase:**
 - The endorsed transaction is sent to the Orderer.
 - The Orderer sequences and packages the transaction into a block.
- **Block Distribution:**
 - The block is distributed to all peers.
 - Each peer writes the updated resident (with the new visitor) into the ledger and updates the state database.
- **Confirmation to Resident:**
 - The system receives confirmation that the transaction was successful.
 - A message is displayed to the resident: `Visitor added successfully`.

3.8.2.5 Block/Unblock Visitor:

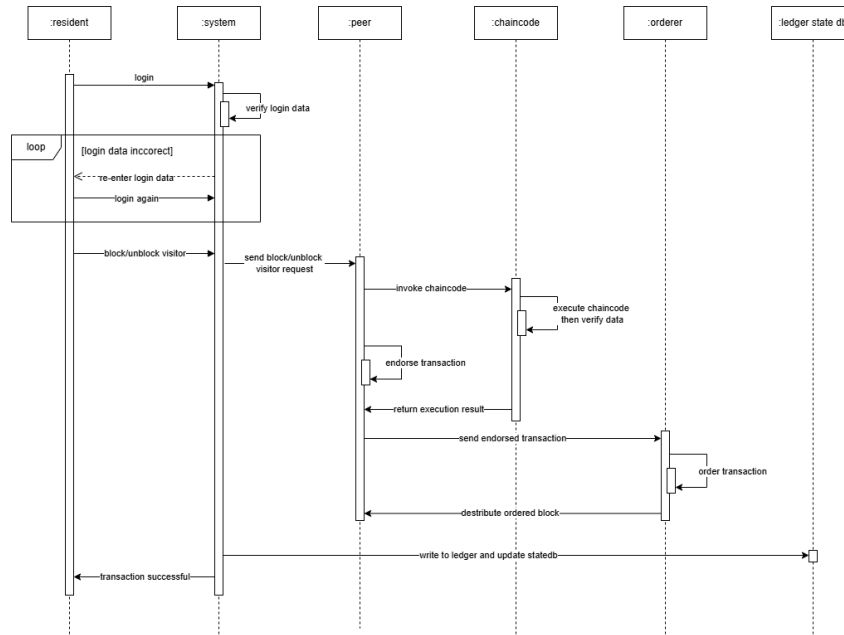


Figure 3.12: Block/Unblock Visitor

- **Block Visitor:**

- The resident selects a visitor and initiates a block request with inputs: `visitorId`, `residentId`, `reason`, `fromDate`, `fromTime`, `toDate`, `toTime`, and `blockedBy`.
- The system sends the block request to the peer node, which invokes the `BlockVisitor` chaincode function.
- The chaincode retrieves the resident data using `GetState("RESIDENT_" + residentId)` and verifies if the visitor exists in the resident's visitor list.
- If the visitor is found and not already blocked:
 - * A `BlockInfo` object is created with the block reason, `blockedBy`, and time range.
 - * The visitors status is set to `"Blocked"` and `CurrentBlock` is set to the new block ID.
 - * The `BlockInfo` is stored in the ledger under a unique key.
- The updated resident record is serialized and stored back in the ledger.
- The transaction is endorsed, ordered by the Orderer, and committed in a new block.
- All peers update the ledger and world state database with the new visitor status and block record.
- The system returns confirmation: `Visitor blocked successfully`.

- **Unblock Visitor:**

- The resident initiates an unblock request by providing the `visitorId` and `residentId`.

- The system sends the request to the peer node, which invokes the **UnblockVisitor** chaincode function.
- The chaincode retrieves the resident and searches for the visitor in their visitor list.
- If the visitor is currently blocked:
 - * The related **BlockInfo** record is deleted using the **CurrentBlock** ID.
 - * The visitors status is updated to "**Active**" and the **CurrentBlock** is cleared.
- The updated resident record is marshaled and stored in the ledger.
- The transaction is endorsed, ordered, and added to the blockchain in a new block.
- Each peer updates the ledger and world state database accordingly.
- The system returns confirmation: **Visitor unblocked successfully.**

3.8.2.6 Update Visitor:

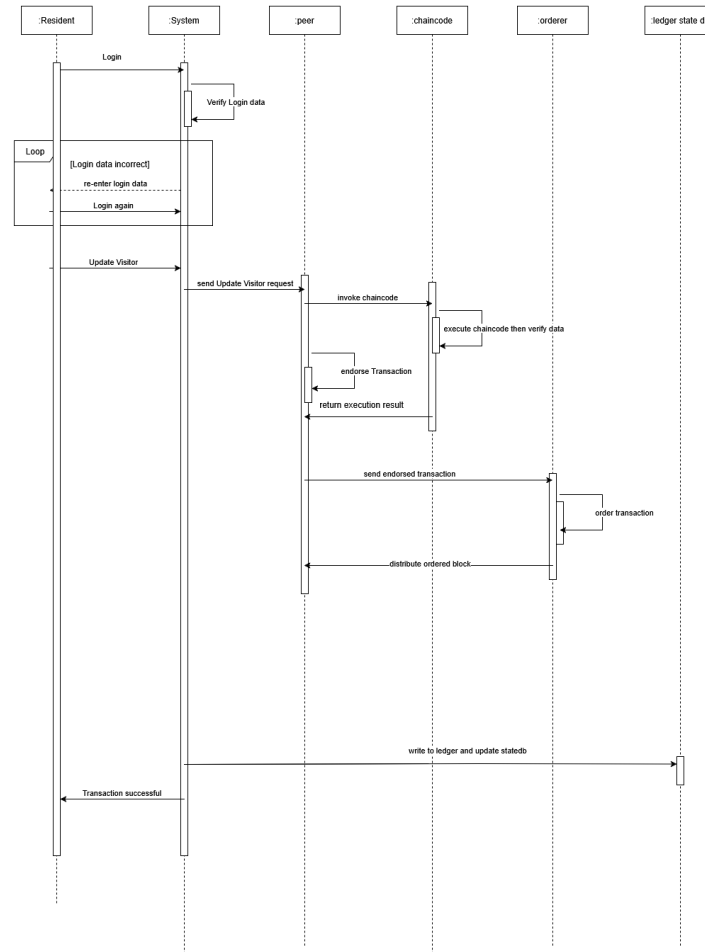


Figure 3.13: Update Visitor

- **Resident/System Initiation:**

- The resident connects to the system and selects a visitor already registered under their profile.
- The resident updates the visitors phone number, visit start time, or end time.
- The resident submits the Update Visitor request.

- **System to Peer:**

- The system sends the update request to the peer node, including the `residentId`, `visitorId`, and updated visitor information.
- The peer invokes the `UpdateVisitor` function in the chaincode.

- **Chaincode Execution:**

- The Admin connects to the system and fills out the visitor request form with details such as: `RequestID`, `VisitorName`, `Phone`, `TargetResident`, `VisitTimeFrom`, `VisitTimeTo`, `VisitDate`, `Type`, and `Purpose`.
- The Admin submits the Add Visit Request targeting a specific resident.
- **System to Peer:**
 - The system sends the request to the peer node in the blockchain network.
 - The peer invokes the `AddVisitRequest` chaincode function with the provided arguments.
- **Chaincode Execution:**
 - The chaincode creates a `VisitRequest` object and sets its status to `Pending`.
 - It serializes the object and stores it in the ledger using the key "`VISITREQUEST_`" + `RequestID`.
- **Endorsement and Ordering Phase:**
 - The peer endorses the transaction and sends the endorsed proposal to the Orderer.
 - The Orderer packages the transaction into a block and distributes it to all peers.
- **Ledger Update:**
 - Each peer writes the new block to the ledger and updates the world state with the new visit request.
- **Confirmation to Admin:**
 - The system receives confirmation of success.

3.8.2.8 Resident approve/reject request of visitor:

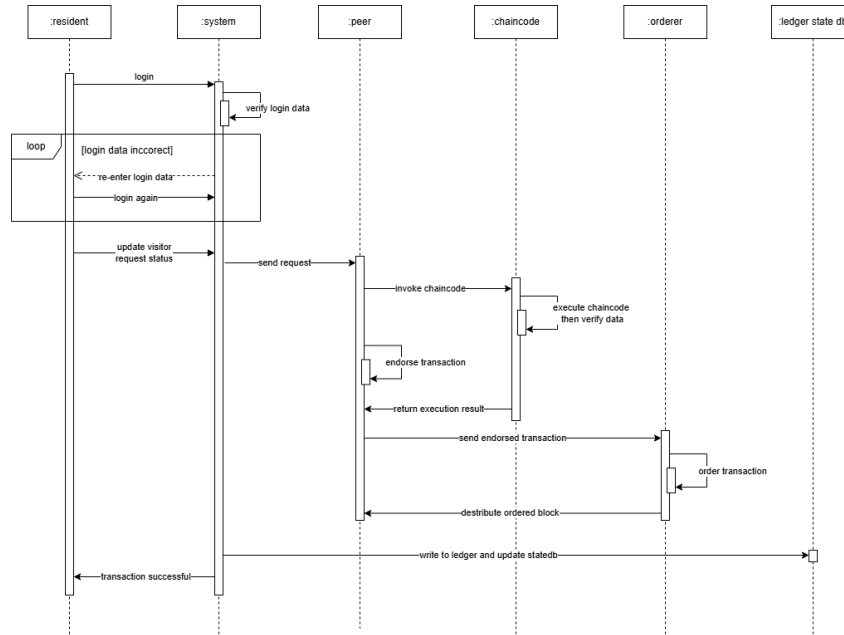


Figure 3.15: approve/reject request of visitor

- **Resident/System Initiation:**

- The resident logs into the system and views pending visit requests submitted by the admin or gatekeeper.
- The resident selects a specific request and chooses to approve or reject it.

- **System to Peer:**

- The system invokes the `UpdateVisitRequestStatus` chaincode function with arguments including `RequestID` and the new `Status` (either "Approved" or "Rejected").

- **Chaincode Execution:**

- The chaincode retrieves the existing `VisitRequest` object using the key "VISITREQUEST_" + `RequestID`.
- It updates the `Status` field based on the residents decision.
- If the status is "Approved", a QR code may be generated and attached to the request.
- The updated request is serialized and prepared for ledger storage.

- **Endorsement and Ordering Phase:**

- The peer endorses the transaction and forwards it to the Orderer.
- The Orderer adds the transaction to a new block and distributes it to all peers.

- **Ledger Update:**

- All peers commit the block and update the world state with the new status of the visit request.
- **Confirmation to Resident and Admin:**
 - The system returns a confirmation that the status has been successfully updated.
 - Both the resident and the admin see the updated status (Approved or Rejected) in their dashboards.

3.9 Conclusion

This chapter provided a comprehensive overview of the system's design. We started by outlining the general architecture of the proposed solution, followed by a detailed explanation of each system component and its role within the Hyperledger Fabric network. We then illustrated the different user interactions and system behaviors using UML diagrams, including use case and sequence. These elements collectively define the systems structure and guide the transition to the next phase. The following chapter will focus on the practical implementation of this design.

Chapter 4

Development and evaluation

4.1 Introduction

In the previous chapter, we presented the overall architecture and detailed design of our proposed system. In this chapter, we focus on the implementation phase. We begin by introducing the tools, technologies, and programming languages used. Then, we describe the implementation of the systems main components with supporting screenshots. Finally, we present and discuss the results obtained from the developed system.

4.2 Software tools

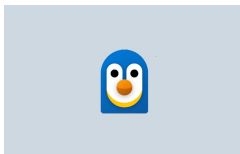
To implement our system, we used a variety of software tools. The development environment was based on VSCode with support from Docker and Docker Compose for containerization. The system was deployed on Ubuntu, and development involved both frontend and backend technologies. The frontend was built using ReactJS and Tailwind-CSS, while the backend used NodeJS, Express, and Go for developing smart contracts. We used Hyperledger Fabric (HLF) as the blockchain platform and CouchDB as the state database. MongoDB Compass was used for managing resident-related data, and Postman was used for testing and validating APIs.



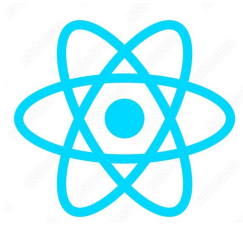
Visual Studio Code: is a free, lightweight code editor available for macOS, Linux, and Windows. It offers a quick and easy setup process, allowing developers to get started within minutes.[\[61\]](#)



Docker: is a platform designed to simplify the development, deployment, and running of applications. It allows you to separate applications from the underlying infrastructure, enabling faster and more consistent software delivery. Using Docker's approach helps streamline processes such as testing and deployment, which reduces the time between writing code and using it in real environments.[\[62\]](#)



WSL: Microsoft created the Windows Subsystem for Linux (WSL) compatibility layer to enable Windows computers to run Linux programs, utilities, and commands through a terminal or command prompt. This conserves computer resources and does away with the requirement for a dual boot computer or virtual machine arrangement. Windows 10 and later versions are required in order to use WSL. Linux distributions like Fedora, Kali, Ubuntu, Arch, and others are supported.[\[63\]](#)



ReactJS: is a JavaScript library for building user interfaces. It enables developers to create reusable UI components, which can be combined to build complex user interfaces for web and native applications. React promotes a component-based architecture, allowing for efficient and flexible development of interactive user interfaces.[64]



Tailwind CSS: is a modern utility-first framework that provides a wide range of low-level CSS classes, allowing developers to create fully customized user interfaces directly within their HTML, without relying on predefined components or styles.[65]



Node.js: is an open-source, cross-platform JavaScript runtime environment that enables the execution of JavaScript code outside of a web browser. It is designed to build scalable network applications by using an event-driven, non-blocking I/O model.[66]



Express: is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It simplifies the development of server-side logic by offering a thin layer of fundamental web application features.[67]



Go: is an open-source programming language designed at Google that makes it easy to build simple, reliable, and efficient software. It is known for its speed, concurrency support, and ease of use.[68]



Hyperledger Fabric: is a modular blockchain platform designed for enterprise use. It supports permissioned networks, where participants have known identities, and allows customization of components such as consensus and membership services. Smart contracts, called chaincode, are used to define business logic.[69]



CouchDB: is an optional external state database for Hyperledger Fabric that allows you to model data on the ledger as JSON and issue rich queries against data values rather than the keys.[70]



MongoDB: is a flexible, document-oriented database designed to simplify application development and scale easily with your data. It stores information in JSON-like documents, giving developers a dynamic, schema-less structure for handling complex data.[71]



Postman: is a comprehensive platform for creating and utilizing APIs. Every phase of the API lifecycle from design and testing to delivery and monitoring is made less painful by it. Postman was created for teams and facilitates collaboration, organization, and the speedier development of safe, dependable APIs.[72]



Hyperledger Fabric Explorer: The Hyperledger Fabric platform's blockchain processes can be visualized with the help of Hyperledger Explorer. Being the first blockchain explorer for permissioned ledgers, it enables anyone to access and examine the distributed ledger projects being developed by Hyperledger members without jeopardizing their privacy. The DTCC, IBM, and Intel all participated to the initiative. Having the ability to visualize data is essential for gaining business value from it. Fortunately, Hyperledger Explorer offers this much-needed feature.[73]



Grafana: is an open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. Commonly used for monitoring metrics in real-time, Grafana supports various databases including Prometheus, InfluxDB, Elasticsearch, and many more.



Prometheus: is an open-source monitoring and alerting toolkit originally built at SoundCloud. It is designed for reliability and scalability, collecting and storing metrics as time series data. Prometheus integrates natively with Grafana, making it a powerful solution for observing systems and applications.

4.3 Hardware Tools

Regarding hardware, our goal was to use NFC cards and NFC readers for secure and fast access control. However, due to the unavailability of physical materials during development, we adopted a temporary solution: using the PC camera to scan QR codes, which acted as digital cards. This workaround allowed us to simulate and test the access control process until the actual hardware becomes available.

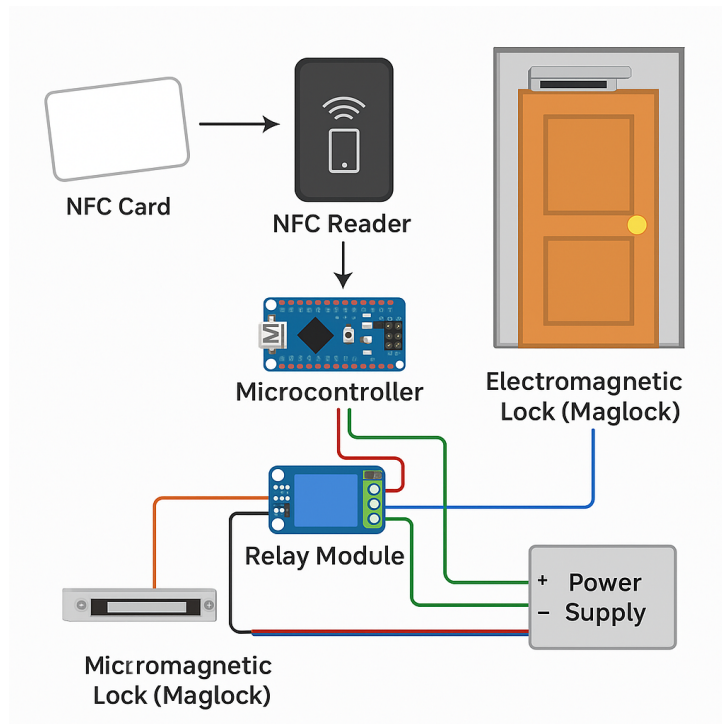


Figure 4.1: Diagram illustrating the typical hardware setup for NFC-based access control.

When the user presents an NFC card to the reader, the microcontroller verifies the ID. If authorized, it triggers the relay module to unlock the electromagnetic lock, allowing access. The system is powered by a 12V DC power supply and ensures quick, secure entry.


Component	Features
	<ul style="list-style-type: none"> • Name: AC/DC Power Adapter • Certification: UL Listed • Compatibility: Specifically designed for Brother AC Adapters (AD24, AD24ES, AD24ESA, AD24A, AD-20, AD-30, AD-60) • Power Specifications: <ul style="list-style-type: none"> – Cord Length: 8.2 feet – Input Voltage Range: 100-240V – Output: 9V, 1.6A (same as original charger) • Safety: Multi-Protect safety system for device and user protection • Supported Models: <ul style="list-style-type: none"> – Brother P-touch models including PT-200G, PT-D210, PTD220, PT-H110, PT-E100, PT-1000, PT-1010 series, PT-1090 series, PT-1200, PT-1230PC, PT-1280 series, PT-1290 series, PT-1300, PT-1400, PT-1500PC, PT-1600, PT-1650, PT-1700, PT-1800, PT-1830 series, PT-1880 series, PT-1890 series, PT-1900, PT-2030 series, PT-2100, PT-2730[74]

Table 4.1: Hardware Components-1


Component	Features
	<ul style="list-style-type: none"> • Name: Arduino Nano V3.0 Development Board • Main Chip: ATmega328P microcontroller • USB Interface Chip: CH340G (not FT232), requires driver installation • Compatibility: Fully compatible with Arduino Nano, Windows, macOS, and Linux • Programming Support: ISP download, USB download • Power Supply Options: <ul style="list-style-type: none"> – Mini-USB connection – 712V unregulated external (pin 30) – 5V regulated external (pin 27) • Power Selection: Automatic selection of highest voltage source (no jumper needed) I/O Pins: <ul style="list-style-type: none"> – 14 digital I/O pins – 6 PWM outputs – 8 analog inputs • Design: Compact, breadboard-friendly, suitable for most applications • USB Standard: Supports full-speed USB, compatible with USB V2.0[75]

Table 4.2: Hardware Components-2


Component	Features
	<ul style="list-style-type: none">• Name: Solenoid Lock• Model: DC 12V Cabinet Door Lock• Rated Operating Voltage: 12V DC• Rated Current: 0.80A• Power Consumption: 9.6Watt• Holding Force: 2.45N• Unlocking Time: 1 sec• Body Material: Iron Metal• Energized forms: Intermittent[76]

Table 4.3: Hardware Components-3


Component	Features
	<ul style="list-style-type: none"> • Name: NFC Reader and Cards • Material: ABS flame-retardant shell • Size: Small and thin • Control: High-quality microchip control for fast card reading • LED Indicator: Red (standby), Green (card reading) • Buzzer: Built-in for effective reminders • Working Frequency: 125 KHz / 13.56 MHz • Card Support: IC / ID dual frequency replication • Interface: USB full speed • USB Cable Length: 1.2 meters • Driver Requirement: None (plug and play) • Response Time: Immediate • Software: Available on connected computer or downloadable via product page[77]

Table 4.4: Hardware Components-4

4.4 Results and discussion

The following results were obtained after executing a high volume of requests over a 15-minute duration. The system under observation was running on a machine equipped with an Intel Core i5-8350U processor and 8 GB of RAM. Performance metrics were collected and visualized using Prometheus for monitoring and Grafana for real-time dashboard analysis. This experimental setup allowed for precise tracking of key indicators such as block processing time, state database commit duration, and overall peer synchronization efficiency:



Figure 4.2: Chaincode Metrics for Peer Requests Over Time

the figure 4.2 shows two graphs focusing on Chaincode Metrics for the Hyperledger Fabric network. The graphs display metrics related to Requests Received and Requests Completed for the chaincode instance `residentManagement` on different peers.

Requests Received

Description: This graph displays the number of incoming requests received by the chaincode instance over time.

Observation: From approximately 08:55 to 09:00, a steady increase in requests is observed, reaching around 37 requests. A sharp drop to zero occurs between 09:00 and 09:02:30, followed by a subsequent rise to approximately 41 requests by 09:05. Another drop to zero appears around 09:09:30, after which the graph shows a gradual increase to about 20 requests by 09:12.

Interpretation: The request pattern reflects intermittent bursts of activity interspersed with pauses, likely indicative of a controlled test scenario involving load injection and idle periods. The peak number of received requests exceeds 40 on two occasions, suggesting the systems capacity to handle such loads effectively.

Requests Completed

Description: This graph illustrates the number of requests successfully processed and completed by the chaincode instance.

Observation: The pattern of completed requests closely mirrors that of the received requests in both shape and magnitude. The graph shows approximately 37 completed requests by 09:00, a drop to zero, a rise to around 41 by 09:05, another drop to zero at 09:09:30, and finally an increase to approximately 20 by 09:12.

Interpretation: The near-identical pattern between received and completed requests strongly indicates that 100% of the received requests were successfully processed, with no

failures, rejections, or visible processing backlogs. This reflects high execution accuracy and operational reliability.

Accuracy: Based on the observed data, the execution accuracy of the system is 100%, as every received request was completed successfully during all periods of activity.

Latency

Observation: The **Requests Completed** graph closely follows the **Requests Received** graph, with no noticeable lag.

Interpretation: The minimal deviation between request reception and completion implies that the system exhibits consistently low latency. The chaincode processes each incoming request almost immediately, indicating efficient throughput and low processing overhead.

Inference

- **Reliability:** No evidence of failures or delays was detected, confirming robust processing.
- **Responsiveness:** Low latency and immediate request handling highlight the systems responsiveness.
- **Accuracy:** The 100% completion rate underscores high accuracy and correctness in processing.
- **Load Handling:** The system adeptly manages fluctuating load conditions without degradation in performance.

Result: Overall Performance

The system demonstrates stable, reliable, and highly accurate performance. It successfully processes all incoming requests with minimal latency and no backlog, even under varying load conditions. This confirms the robustness and scalability of the Hyperledger Fabric network during the evaluated test window.

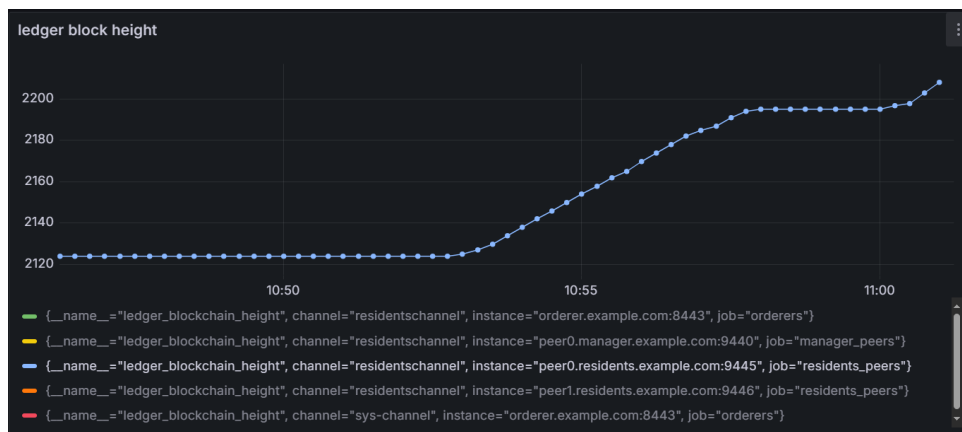


Figure 4.3: Ledger Blockchain Height Metric for Resident Peer

The figure 4.3 displays a graph illustrating the `ledger_blockchain_height` metric for a resident peer (`peer0.residents.example.com:9445`) on the `residentschannel` in a Hyperledger Fabric network.

Throughput

Observation: The blockchain height increases from approximately 2120 to 2205 over an 8-minute period (from 10:52 to 11:00). This represents an increase of approximately 85 blocks in 8 minutes.

Interpretation: The observed block generation rate for this resident peer is approximately 85 blocks/8 minutes 10.625 blocks/minute, or roughly 0.18 blocks/second. This indicates a moderate and consistent throughput of transactions being processed and committed to the ledger on the `residentschannel` during the active period. The ability of the peer to consistently update its ledger at this rate demonstrates its effective participation in the network's block synchronization and validation process.

Result

The `ledger_blockchain_height` graph for the resident peer clearly shows that after an initial period of inactivity, the Hyperledger Fabric network began actively committing transactions to the `residentschannel`. The consistent increase in block height signifies that this specific resident peer is successfully receiving and validating new blocks, maintaining an up-to-date ledger. The calculated throughput confirms a steady rate of block generation and processing on this channel, indicating healthy and synchronized operation during the observed active period. **StateDB Commit Time**

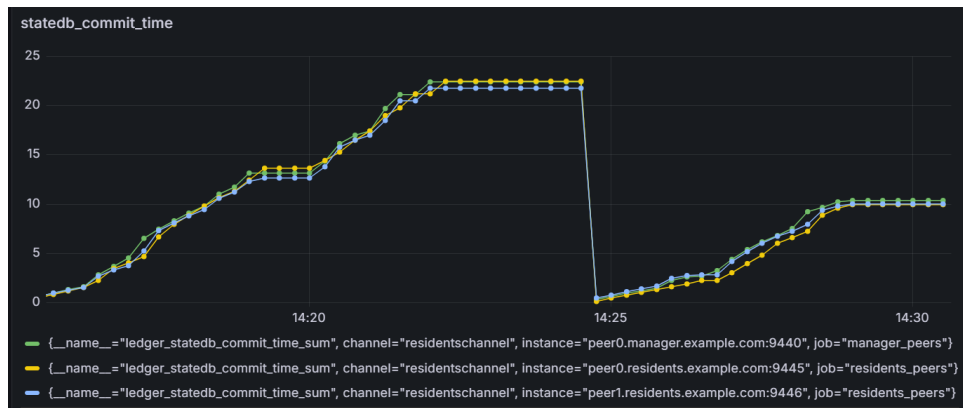


Figure 4.4: StateDB Commit Time Metric for Peers on `residentschannel`

The figure 4.4 displays the cumulative time taken to commit changes to the StateDB database (`ledger_statedb_commit_time_sum`) across three peers in a Hyperledger Fabric network.

Observations

Peers exhibited steady commit time growth until approximately 14:22, indicating smooth and consistent transaction processing. This was followed by a stable phase from

14:22 to 14:24, where commit times plateaued, suggesting minimal new activity but efficient ongoing operations. Around 14:24, a synchronized drop in metrics across all peers likely points to a planned reset or maintenance event. Notably, the peers resumed normal behavior immediately afterward, demonstrating the networks robustness and ability to recover quickly.

Result

- StateDB commits efficiently handle transaction loads.
- Peers maintain synchronization during both active and idle periods.
- The system recovers seamlessly from interruptions, ensuring consistent state updates.

Block Processing Time



Figure 4.5: ledger block processing time

The figure 4.5 displays the cumulative time taken for `ledger_block_processing_time_sum` across three peers in a Hyperledger Fabric network on the `residentschannel`. This metric typically represents the total time spent by a peer in processing blocks received from the ordering service, which includes validation, execution of transactions, and committing to the ledger and state database.

Observations

The graph illustrates strong, synchronized performance across all three peers on the `residentschannel` for the `ledger_block_processing_time_sum` metric. Initially (18:12–~18:16), peers showed steady growth in processing time, indicating efficient transaction flow. Minor timing differences between peers reflect normal operational variance. Between ~18:16 and ~18:18:30, the metric plateaued, signaling stable processing without backlog. A sharp, coordinated drop around ~18:18:30 suggests a successful, synchronized reset. Following this, from ~18:19 to 18:25, peers resumed steady growth, confirming the networks resilience and efficient recovery.

Result

- **Resilient and Efficient Network:** The graph demonstrates a highly robust system performing optimally under varying conditions.
- **Consistent Block Processing:**
 - Steady growth phases confirm reliable transaction processing
 - Peers maintain synchronization during active periods
- **Efficient Load Handling:**
 - Smooth management of variable transaction volumes
 - Stable performance during both active and plateau phases
- **Coordinated System Reset:**
 - Synchronized behavior across all peers during reset
 - Demonstrates strong network-wide coordination
- **Rapid Recovery Capability:**
 - Immediate resumption of normal operations post-reset
 - Maintains optimal performance levels after interruptions
- **Overall Assessment:**
 - Well-orchestrated Hyperledger Fabric network
 - Demonstrates high-level performance characteristics
 - Robust architecture capable of handling operational variations

4.5 Implementation of the System

4.5.1 Login

The login process for the HexiBuilding system follows a role-based authentication mechanism to distinguish between administrators and residents. The steps are as follows:

- **User Opens Login Page:** The user (admin or resident) accesses the HexiBuilding login interface.
- **User Enters Credentials:** The user provides their email address and password, then clicks the *Login* button.
- **Backend Authentication:** The system verifies the submitted credentials against the database. If the credentials are invalid, an error message such as *"Invalid credentials"* is displayed.

- **Role Identification:** Upon successful authentication, the system retrieves the user's role:
 - If the user is an administrator, the role is identified as "*admin*".
 - If the user is a resident, the role is identified as "*resident*".
- **Redirection Based on Role:** The system redirects the user based on their role:
 - If role = *admin* → redirect to the **Admin Dashboard**.
 - If role = *resident* → redirect to the **Resident Dashboard**.

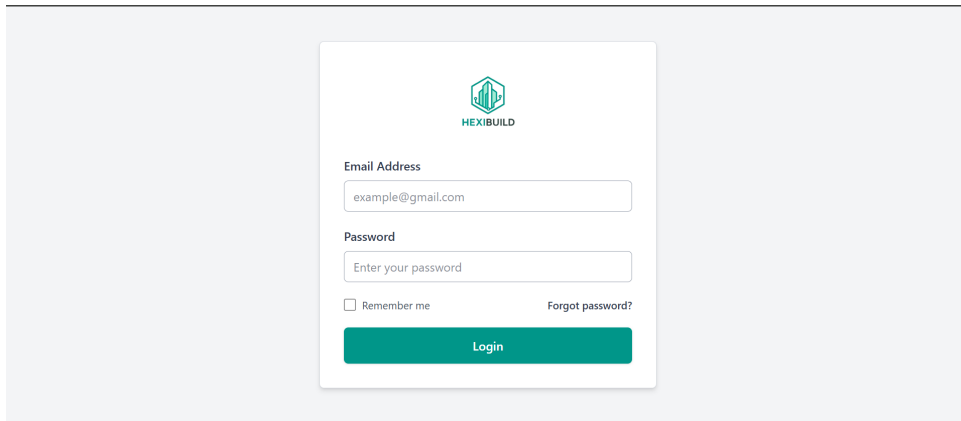


Figure 4.6: Login Interface

4.5.2 Dashboard of Admin

4.5.2.1 Overview:

Admin sees a dashboard overview with:

- Total number of residents
- Total apartments
- Total messages
- Visitor Attendance

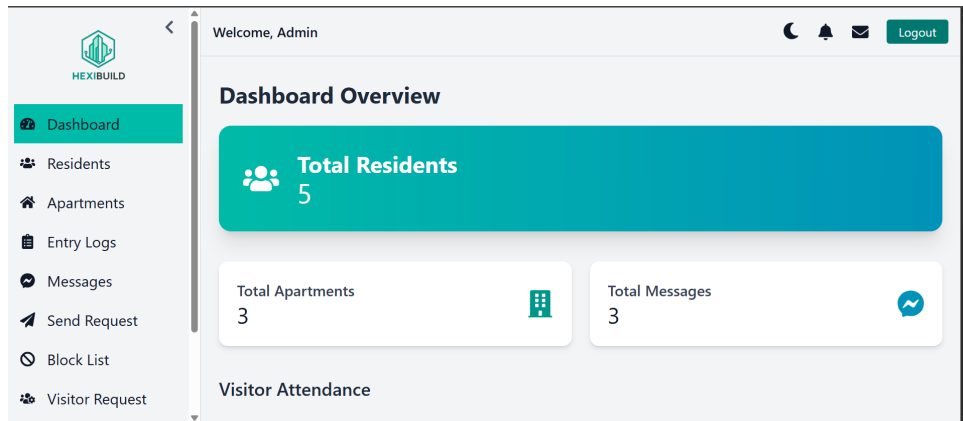


Figure 4.7: Dashboard Overview

4.5.2.2 Manage Apartments:

The admin is responsible for managing apartment data within the system. This includes the ability to add new apartments by specifying relevant details such as the apartment Name, Description. In addition, the admin can update existing apartment information to reflect changes or correct errors, and also has the authority to delete apartments when they are no longer in use or relevant. These functionalities ensure that the apartment records are accurate, up-to-date, and aligned with the current status of the building.

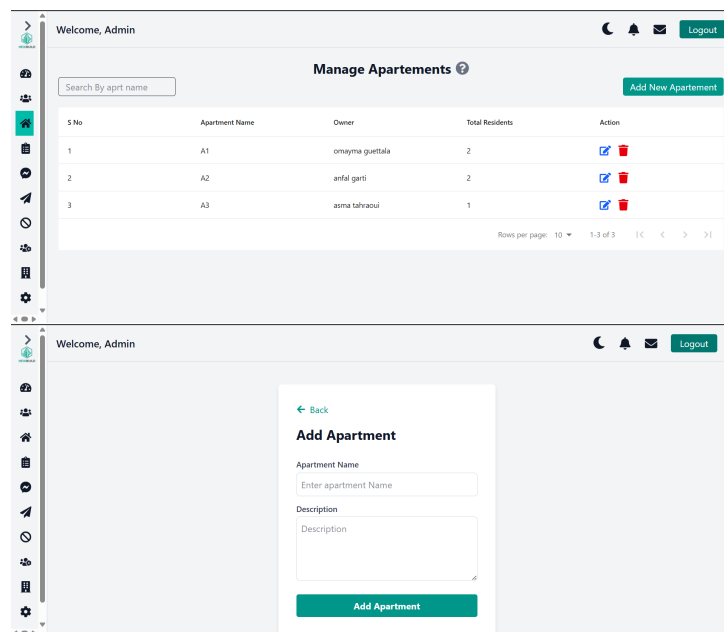


Figure 4.8: Manage Appartments

4.5.2.3 Manage Residents:

The admin has full control over resident management within the system. This includes the ability to add new residents by entering their personal and apartment-related information, as well as updating or deleting resident records when necessary. Additionally, the admin

can block or unblock residents based on policy violations or access requirements. When a new resident is added, the system automatically generates a unique QR code that allows them to access the building securely. Similarly, the admin can manage each residents list of approved visitors adding new visitors, editing details, blocking or removing them and for each approved visitor, the system dynamically generates a QR code to facilitate secure entry. This comprehensive management ensures that access is controlled and in full compliance with the buildings security protocols.

The screenshot shows the 'Manage Residents' interface. On the left is a sidebar with navigation options: Dashboard, Residents (selected), Apartments, Entry Logs, Messages, Send Request, Block List, Visitor Request, Building, and Settings. The main content area is titled 'Manage Residents' and includes a search bar 'Search By Res name' and an 'Add New Resident' button. Below this is a table with the following data:

S No	Name	Image	Contact Info	Apartments	Type	Status	Action
1	asma tah		056999333 asmatahraoui@gmail.com	Apartment 2	owner	Active	
2	Anfal		055142638577 anfalqumr@gmail.com	Apartment 5	family	Blocked	
3	abir		0551526588 abir123@gmail.com	Apartment 3	owner	Active	
4	Ahmed		0771589744 ahmed@gmail.com	Apartment 4	family	Active	

At the bottom right of the table, there is a pagination control showing 'Rows per page: 10', '1-4 of 4', and navigation arrows.

Figure 4.9: Manage Residents

4.5.2.4 Manage Requests:

The admin can initiate a visitor access request to a resident by submitting the visitors information. This request is then sent to the respective resident for approval. Once the resident responds, the system automatically updates the request status from "pending" to either "accepted" or "rejected" based on the residents decision. If the request is accepted, the system securely generates a unique QR code for the visitor, granting them authorized entry during the specified access period.

Send Visit Request

Apartment: Select Apartment Residents: Select Resident

Visitor Name: Insert Name Phone: Phone

Visit Type: Single Visit Purpose: Personal

Custom Reason: Reason Visit Date: j/m/aaaa

Visit Time From: --:-- Visit Time To: --:--

Send Request

Welcome, Admin Logout

Manage Requests

Search By Visitor Name

S No	Name	type	Purpose	Reason	TimeFrom	TimeTo	Status	Action
1	manel slimani	single	personal	old neighbor want...	19:08	20:08	accepted	No action
2	Kahina hzarta	single	personal	old friend want to...	21:17	22:17	pending	No action
3	Khalil saidi	single	delivery	Food delivery	12:15	12:45	accepted	No action
4	Mohamed Ali	group	maintena...	elevator maintena...	10:00	14:00	accepted	No action
5	Abir Iitim	single	personal	old neighbor want...	10:44	10:46	pending	No action
6	nouha haddadi	single	personal	old neighbor want...	10:30	11:20	pending	No action
7	mira ben abdelk...	single	personal	old friend want to...	19:27	19:30	accepted	No action

Request Details

Visitor Name: Khalil saidi

Visitor Phone: 0657283456

Request type: single

Visit Purpose: delivery

Reason: Food delivery

Time From: 12:15

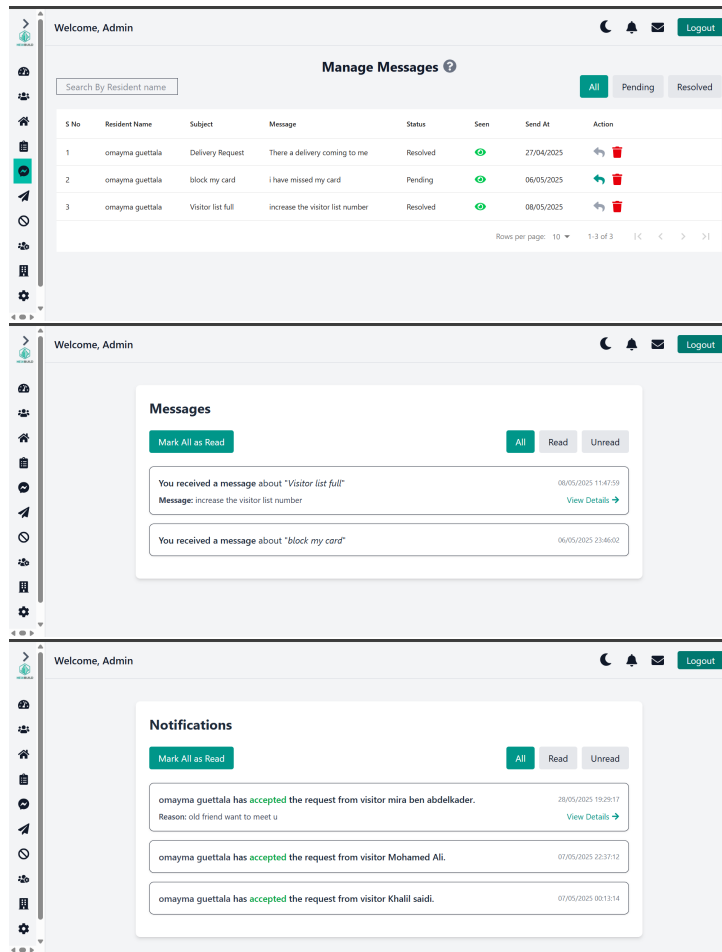
Time To: 12:45

Status: accepted

Figure 4.10: Manage Requests

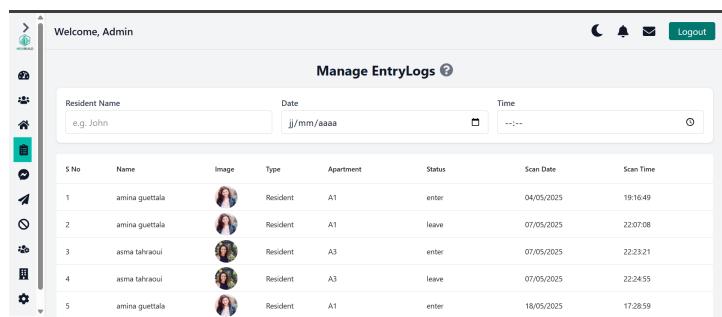
4.5.2.5 Notifications/ Manage Messages:

Admin receive notifications and also manage messages.



4.5.2.6 Manage EntryLogs:

This page displays real-time records of residents and visitors entering or leaving the building. It provides detailed logs including entry and exit timestamps, user identity (resident or visitor), and access method used (QR code) like date of scan and time of scan. This tracking mechanism enhances security by maintaining a transparent and auditable history of all access events within the building.



4.5.2.7 Manage Block List:

If the admin blocks a resident, this page provides a detailed view of all blocked residents. It displays comprehensive information including the name and reason and from date to

date and also from time to time...etc. This feature ensures accountability and helps administrators monitor and manage restricted access cases effectively.

S No	Name	Type	Reason	Blocked By	From Date	From Time	To Date	To Time	Action
1	amina guettala	Resident	Card missed	Admin	08/06/2025	20:10:00	13/06/2025	20:10:00	

4.5.3 Resident Dashboard

4.5.3.1 Manage Profile:

omayma guettala
ID: RES-970394

Phone: 0657283978 | Email: omaymaguettala@gmail.com

Gender: female | Apartment: A1

Marital Status: single | Resident Type: owner

[Change Password](#)

4.5.3.2 Manage Visitors:

Each resident has the ability to manage their personal list of visitors within the system. When a resident registers a new visitor, the system automatically generates a dynamic QR code that grants the visitor access to the building. Additionally, residents have full control over their visitor list, including the ability to edit visitor details, block specific individuals, or remove them entirely. This functionality empowers residents to maintain secure and personalized access for their guests.

Welcome, omayma guettala

← Back

Add New Visitor

Name Phone

Visit Time From Visit Time To

Relationship

4.5.3.3 Manage Requests:

When a resident receives an access request from the admin typically for a visitor the resident has the option to either accept or reject the request. If the resident accepts it, the system automatically generates a QR code for the visitor. This QR code serves as a secure, time-sensitive credential that allows the visitor to enter the building, ensuring controlled and authorized access.

Welcome, omayma guettala

Search By Res name

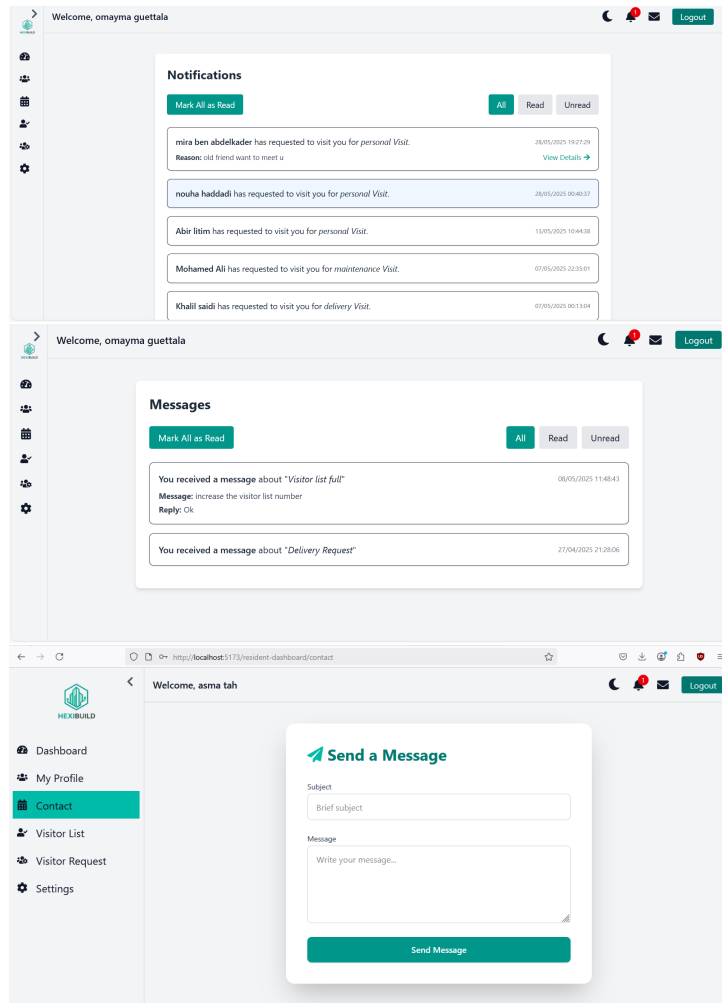
Manage Requests

S No	Name	type	Purpose	Reason	Timefrom	TimeTo	Status	Action
1	marouf elmani	single	personal	old neighbor want...	19:08	20:08	accepted	<input checked="" type="checkbox"/> <input type="checkbox"/>
2	Khalil sadi	single	delivery	Food delivery	12:15	12:45	accepted	<input checked="" type="checkbox"/> <input type="checkbox"/>
3	Mohamed Ali	group	maintena...	elevator maintena...	10:00	14:00	accepted	<input checked="" type="checkbox"/> <input type="checkbox"/>
4	Abir Itim	single	personal	old neighbor want...	10:44	10:46	pending	<input type="checkbox"/> <input checked="" type="checkbox"/>
5	nouha haddadi	single	personal	old neighbor want...	10:30	11:20	pending	<input type="checkbox"/> <input checked="" type="checkbox"/>
6	mira ben abdelk...	single	personal	old friend want to ...	19:27	19:30	accepted	<input checked="" type="checkbox"/> <input type="checkbox"/>

Rows per page: 10 1-6 of 6

4.5.3.4 Notifications/Messages:

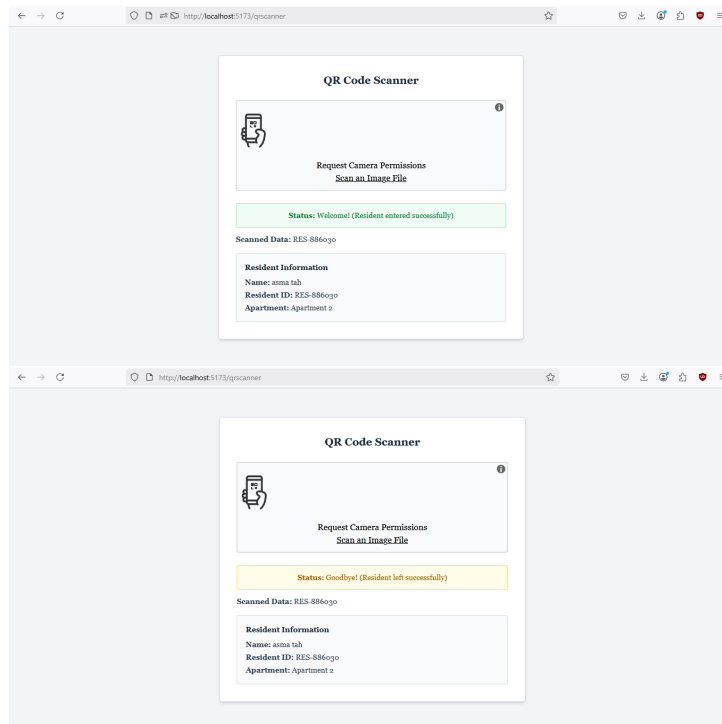
Resident receive notifications and send/receive messages to/from admin.



4.5.4 Scan Page

This page is responsible for scanning and verifying the QR code information of Residents, Registered Visitors, and Requested Visitors.

- For Residents, access is granted if the QR code ID is valid and the resident is not blocked.
- For Registered Visitors, the system ensures that the QR code is valid, the visitor is not blocked, and the code has not expired before granting access.
- For Requested Visitors, the QR code must be valid, the scheduled visit date must match the current date, and the code must still be within its validity period. If all these conditions are satisfied, the visitor is allowed entry into the building.



4.5.5 Guide

Each page in the system includes a help icon ("?"). When a user clicks this icon, a brief explanation is displayed to guide them on how the page functions and how to use its features effectively. This ensures a more user-friendly experience by providing immediate assistance and improving system accessibility.

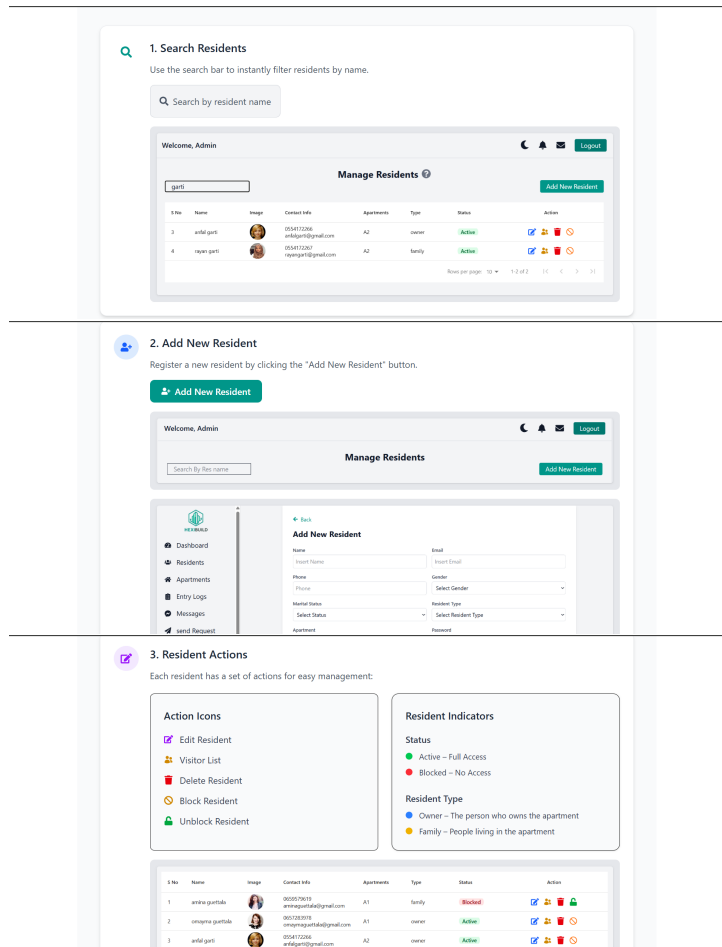


Figure 4.11: Exemple the guide in Manage Residents page

4.6 Conclusion

This chapter presented the implementation of the proposed system, covering the tools used and key functional modules. The results demonstrated the systems reliability, responsiveness, and effective load handling, confirming that it meets the design goals and is ready for further development or deployment.

General Conclusion

Access control in smart buildings has become a critical challenge as IoT and smart technologies continue to evolve. Traditional systems, which rely on centralized databases and static permissions, are prone to security breaches, lack transparency, and struggle with scalability. These inefficiencies highlight the need for a more secure, decentralized, and auditable solution.

In this thesis, we proposed a blockchain-based access control system using Hyperledger Fabric to address these challenges. Our solution eliminates single points of failure by decentralizing control, ensures transparency through immutable logs of access events, and improves scalability to accommodate growing numbers of users and access points.

Through this project, we designed and developed a secure, tamper-proof access control framework that enhances security and accountability in smart buildings. The system provides real-time access management, automated permission updates, and a verifiable audit trail all while maintaining high performance.

As future work, we aim to expand the application of blockchain technology beyond access control by integrating a property management module for buying, selling, and renting apartments within smart buildings. This extension would leverage smart contracts to automate transactions, ensure trustless agreements, and maintain a transparent ledger of ownership and rental history.

By doing so, we can create a fully decentralized smart building ecosystem that enhances security, efficiency, and user convenience in all aspects of building management.

Bibliography

- [1] Somia Sahraoui. “Mécanismes de sécurité pour l'intégration des RCSFs à IIoT (Internet of Things).” PhD thesis. Université de Batna 2, 2016.
- [2] Ruchit Parekh. “Automating the design process for smart building technologies.” In: *World Journal of Advanced Research and Reviews* 23.2 (2024).
- [3] Leepakshi Bindra et al. “Flexible, decentralised access control for smart buildings with smart contracts.” In: *Cyber-Physical Systems* 8.4 (July 2021), pp. 286–320. ISSN: 2333-5785. DOI: [10.1080/23335777.2021.1922502](https://doi.org/10.1080/23335777.2021.1922502). URL: <http://dx.doi.org/10.1080/23335777.2021.1922502>.
- [4] ServiceChannel. *7 Smart Building Technologies*. Accessed on 26th April, 2025. 2025. URL: <https://servicechannel.com/blog/7-smart-building-technologies/>.
- [5] Mid-Atlantic Controls. *Benefits of Smart Building Management Systems*. Accessed on 26th April, 2025. 2025. URL: <https://info.midatlanticcontrols.com/blog/benefits-of-smart-building-management-systems>.
- [6] U.S. Department of Energy. *Wireless Occupancy Sensors for Lighting Controls: An Applications Guide for Federal Facility Managers*. Pages 1--3. 2016. URL: <https://www.energy.gov/femp/articles/wireless-occupancy-sensors-lighting-controls-applications-guide-federal-facility>.
- [7] MEP Academy. *How Occupancy and Vacancy Sensors Work*. Accessed on 27th April, 2025. 2022. URL: <https://mepacademy.com/how-occupancy-and-vacancy-sensors-work/>.
- [8] Renkeer. *Types of Smart Building Sensors With IoT Technology*. Accessed on 27th April, 2025. 2025. URL: <https://www.renkeer.com/smart-building-sensors-types/>.
- [9] Designing Buildings Wiki. *Building Energy Management Systems (BEMS)*. Accessed on 26th April, 2025. 2025. URL: https://www.designingbuildings.co.uk/wiki/Building_energy_management_systems.
- [10] Archova Visuals. *The Future of IT and Communications Infrastructure in Smart Commercial Buildings*. Accessed on 27th April, 2025. 2025. URL: <https://archovavisuals.com/communications-in-smart-commercial-buildings/>.
- [11] Beatrice Witzgall and Andy McMillan. *BACnet Provides an Ideal Smart Building Backbone*. Accessed: 2025-05-15. Oct. 2021. URL: <https://www.buildings.com/building-systems-om/lighting/article/55257368/bacnet-provides-an-ideal-smart-building-backbone-magazine>.

- [12] DesignHorizons. *IoT in Modern Buildings: Key Components and Applications*. Accessed on 27th April, 2025. 2025. URL: <https://designhorizons.org/iot-in-modern-buildings-key-components-and-applications/>.
- [13] Francisco-Javier Ferrández-Pastor et al. “Deployment of IoT Edge and Fog Computing Technologies to Develop Smart Building Services.” In: *Sustainability* 10.11 (2018), p. 3832. DOI: [10.3390/su10113832](https://doi.org/10.3390/su10113832). URL: <https://www.mdpi.com/2071-1050/10/11/3832>.
- [14] Architecture Industry. *The Evolution of Smart Buildings and IoT in Architecture*. Accessed on 27th April, 2025. 2025. URL: <https://architectureindustry.in/the-evolution-of-smart-buildings-and-iot-in-architecture-202209261050/>.
- [15] Mervi Himanen. “The Significance of User Involvement in Smart Buildings Within Smart Cities.” In: Dec. 2017, pp. 265–314. ISBN: 978-3-319-44922-7. DOI: [10.1007/978-3-319-44924-1_13](https://doi.org/10.1007/978-3-319-44924-1_13).
- [16] Intelligent Buildings. *Maximizing Resource Utilization in Your Smart Building*. Accessed: April 29, 2025. 2025. URL: <https://intelligentbuildings.com/maximizing-resource-utilization-in-your-smart-building/>.
- [17] S. Bhatia and M. Saini. “IoTA Promising Solution to Energy Management in Smart Buildings: A Systematic Review, Applications, Barriers, and Future Scope.” In: *Buildings* 14.11 (2024), p. 3446. DOI: [10.3390/buildings14113446](https://doi.org/10.3390/buildings14113446). URL: <https://doi.org/10.3390/buildings14113446>.
- [18] Security Journal Americas. *Transforming Multifamily Living with Intelligent Buildings*. Accessed: April 29, 2025. 2023. URL: <https://securityjournalamericas.com/multifamily-intelligent-buildings/>.
- [19] Eseye. *IoT in Smart Buildings: Foundations for Success*. Accessed: April 29, 2025. 2023. URL: <https://www.eseye.com/resources/blogs/iot-in-smart-buildings-foundations-for-success/>.
- [20] Dom Beveridge and Donald Davidoff. *Access Control and the Future of Smart Buildings*. Tech. rep. Accessed: April 29, 2025. D2 Demand Solutions, 2021. URL: https://20for20.com/wp-content/uploads/2022/04/Apr21_D2_Latch_Access-Control_White-Paper.pdf.
- [21] Nian Xue et al. “An Access Control System for Intelligent Buildings.” In: ACM, Dec. 2016. DOI: [10.4108/eai.18-6-2016.2264493](https://doi.org/10.4108/eai.18-6-2016.2264493).
- [22] Scott Harper et al. “User Privacy Concerns in Commercial Smart Buildings.” In: *Journal of Computer Security* 30.3 (2022), pp. 241–273. DOI: [10.3233/JCS-210035](https://doi.org/10.3233/JCS-210035). URL: <https://journals.sagepub.com/doi/10.3233/JCS-210035>.
- [23] Vincent Hu, D. Kuhn, and David Ferraiolo. “Attribute-Based Access Control.” In: *Computer* 48 (Feb. 2015), pp. 85–88. DOI: [10.1109/MC.2015.33](https://doi.org/10.1109/MC.2015.33).
- [24] Axiomatics. *Intro to Attribute-Based Access Control (ABAC)*. Accessed: April 29, 2025. 2023. URL: <https://axiomatics.com/blog/intro-to-attribute-based-access-control-abac/>.
- [25] Daniela Popescu and Marcela Prada. “Some Aspects about Smart Building Management Systems -Solutions for Green, Secure and Smart Buildings.” In: Mar. 2013. DOI: [10.13140/RG.2.1.3057.8644](https://doi.org/10.13140/RG.2.1.3057.8644).

- [26] Amir Sekhavat et al. *Navigating the AI Revolution in Smart Buildings: Opportunities, Roadblocks, and Strategies*. Accessed: May 1, 2025. 2025. URL: <https://tiaonline.org/navigating-the-ai-revolution-in-smart-buildings-opportunities-roadblocks-and-strategies/>.
- [27] June Young Park et al. "A critical review of field implementations of occupant-centric building controls." In: *Building and Environment* 165 (2019), p. 106351. DOI: [10.1016/j.buildenv.2019.106351](https://doi.org/10.1016/j.buildenv.2019.106351). URL: <https://www.sciencedirect.com/science/article/pii/S036013231930561X>.
- [28] Imanol Martín Toral et al. "Introducing Security Mechanisms in OpenFog-Compliant Smart Building Applications." In: *Electronics* 13.15 (2024), p. 2900. DOI: [10.3390/electronics13152900](https://doi.org/10.3390/electronics13152900). URL: <https://www.mdpi.com/2079-9292/13/15/2900>.
- [29] Oak Tree Technologies. *An Overview of Hyperledger: Understanding the Framework*. Accessed on 29th April, 2025. 2025. URL: <https://www.oak-tree.tech/blog/hyperledger-overview>.
- [30] Peter Gratton. *What Is a Block in the Crypto Blockchain, and How Does It Work?* Accessed: April 30, 2025. 2025. URL: <https://www.investopedia.com/terms/b/block-bitcoin-block.asp>.
- [31] MoonPay. *What are Blockchain Nodes?* Accessed: April 30, 2025. 2025. URL: <https://www.moonpay.com/learn/blockchain/what-are-blockchain-nodes>.
- [32] UpGrad. *What is a Blockchain Transaction?* Accessed: April 30, 2025. 2025. URL: <https://www.upgrad.com/blog/what-is-blockchain-transaction/>.
- [33] Somia Sahraoui and Abdelmalik Bachir. "Lightweight Consensus Mechanisms in the Internet of Blockchained Things: Thorough Analysis and Research Directions." In: *Digital Communications and Networks* (2025).
- [34] Unknown. *Major components of blockchain*. Accessed: 2025-05-04. 2022. URL: https://www.researchgate.net/figure/Major-components-of-blockchain_fig1_357950461.
- [35] Amazon Web Services. *What is Blockchain?* Accessed: April 29, 2025. 2025. URL: <https://aws.amazon.com/what-is/blockchain/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc>.
- [36] Henry Koko. *Blockchain Technology: Discover 5 Unique Features of Blockchain Technology!* Accessed: April 30, 2025. 2024. URL: <https://cryptopulseblog.com/blockchain-technologydiscover-5-unique-features-of-blockchain-technology/>.
- [37] Casey Lindner. *The Difference Between Encryption, Hashing, and Salting*. Accessed: 2025-05-04. 2019. URL: <https://www.thesslstore.com/blog/difference-encryption-hashing-salting/>.
- [38] Arthur D. Little. *How Blockchain Platforms Enhance Telecom & Media*. Accessed: 2025-05-04. 2022. URL: <https://www.adlittle.com/en/insights/viewpoints/how-blockchain-platforms-enhance-telecom-media>.

- [39] Rapid Innovation. *Consensus Mechanisms in Blockchain: Proof of Work vs. Proof of Stake and Beyond*. Accessed: April 30, 2025. 2025. URL: <https://www.rapidinnovation.io/post/consensus-mechanisms-in-blockchain-proof-of-work-vs-proof-of-stake-and-beyond>.
- [40] Dave Patten. *Consensus Mechanisms Explained: PoW, PoS, and Beyond*. Accessed: 2025-05-04. 2025. URL: <https://coinsbench.com/consensus-mechanisms-explained-pow-pos-and-beyond-9121c8ad9eea>.
- [41] bandonkeyea_78374. *Upgradable Smart Contracts in Solidity: A Comprehensive Guide*. Accessed: 2025-05-04. 2025. URL: https://medium.com/@bandonkeyea_78374/upgradable-smart-contracts-in-solidity-a-comprehensive-guide-e5627c896ea4.
- [42] GeeksforGeeks. *How Does the Blockchain Work?* Accessed: 2025-05-04. 2021. URL: <https://www.geeksforgeeks.org/how-does-the-blockchain-work/>.
- [43] GeeksforGeeks. *Blockchain Structure*. Accessed on 29th April, 2025. 2025. URL: <https://www.geeksforgeeks.org/blockchain-structure/>.
- [44] Hyperledger Fabric Contributors. *What is Hyperledger Fabric?* Accessed: 2025-05-01. 2024. URL: <https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html>.
- [45] Hyperledger Fabric Contributors. *Membership Service Provider (MSP)*. Accessed: 2025-05-01. 2024. URL: <https://hyperledger-fabric.readthedocs.io/en/release-2.5/membership/membership.html>.
- [46] Hyperledger Fabric Documentation. *Using CouchDB*. Accessed: 2025-05-01. 2024. URL: https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_tutorial.html.
- [47] Hyperledger Fabric Contributors. *Endorsement Policies*. Accessed: 2025-05-01. 2024. URL: <https://hyperledger-fabric.readthedocs.io/en/release-2.5/endorsement-policies.html>.
- [48] AST Consulting. *Understanding Channels in Hyperledger Fabric*. Accessed: 2025-05-01. 2023. URL: <https://astconsulting.in/blockchain/hyperledger-fabric/understanding-channels-hyperledger>.
- [49] Hyperledger Fabric Contributors. *Chaincode* Hyperledger Fabric Documentation. Accessed: 2025-05-01. 2019. URL: <https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html>.
- [50] AST Consulting. *Membership Service Providers in Hyperledger Fabric*. Accessed: 2025-05-01. 2023. URL: <https://astconsulting.in/blockchain/membership-service-providers-in-hyperledger-fabric>.
- [51] Hyperledger Fabric Documentation. *Hyperledger Fabric: Blockchain Network*. Accessed: 2025-05-04. 2022. URL: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/network/network.html>.
- [52] Hyperledger Fabric Documentation. *Creating a Channel* Hyperledger Fabric Documentation. Accessed: 2025-05-04. 2022. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/create_channel/create_channel_overview.html.

- [53] Hyperledger Fabric Documentation. *Ordering Service Hyperledger Fabric Documentation*. Accessed: 2025-05-04. 2022. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html.
- [54] IBM Documentation. *Deploy a smart contract*. <https://www.ibm.com/docs/en/hlf-support/1.0.0?topic=contracts-deploy-smart-contract>. Accessed: 2025-05-01. 2023.
- [55] IBM Documentation. *Hyperledger Fabric Reference*. <https://www.ibm.com/docs/en/blockchain-platform/2.5.2?topic=reference-hyperledger-fabric>. Accessed: 2025-05-01. 2024.
- [56] IBM Corporation. *What is Hyperledger?* Accessed: 2025-05-04. 2024. URL: <https://www.ibm.com/think/topics/hyperledger>.
- [57] Kaleido. *10 Use Cases for Hyperledger Fabric*. Accessed: 2025-05-04. 2023. URL: <https://www.kaleido.io/blockchain-blog/10-use-cases-for-hyperledger-fabric>.
- [58] Lakhveer Bindra et al. “Flexible, Decentralized Access Control for Smart Buildings with Smart Contracts.” In: *arXiv preprint arXiv:2010.08176* (2020). URL: <https://arxiv.org/abs/2010.08176>.
- [59] NineSmart. *Smart Access / QR Code Access Control*. <https://ninesmart.io/smart-access/>. Accessed: 2025-06-05.
- [60] Jian Wu et al. “SPCL: A Smart Access Control System That Supports Blockchain.” In: *Applied Sciences* 14.7 (2024), p. 2978. DOI: [10.3390/app14072978](https://doi.org/10.3390/app14072978). URL: <https://doi.org/10.3390/app14072978>.
- [61] Visual Studio Code. *Setup Overview*. Accessed: 2025-05-22. 2025. URL: <https://code.visualstudio.com/docs/setup/setup-overview>.
- [62] Docker Inc. *Docker Overview*. Accessed: 2025-05-22. 2025. URL: <https://docs.docker.com/get-started/docker-overview/>.
- [63] GeeksforGeeks. *What is Windows Subsystem for Linux (WSL)?* Accessed: 2025-05-23. 2025. URL: <https://www.geeksforgeeks.org/what-is-windows-subsystem-for-linux-wsl/#what-is-windows-subsystem-for-linux-wsl->.
- [64] Meta Platforms Inc. *React Official Website*. Accessed: 2025-05-22. 2025. URL: <https://react.dev/>.
- [65] Tailwind Labs. *Tailwind CSS Official Website*. Accessed: 2025-05-22. 2025. URL: <https://tailwindcss.com/>.
- [66] OpenJS Foundation. *About Node.js*. Accessed: 2025-05-22. 2025. URL: <https://nodejs.org/en/about>.
- [67] OpenJS Foundation. *Express - Node.js Web Application Framework*. Accessed: 2025-05-22. 2025. URL: <https://expressjs.com/>.
- [68] Go Developers. *The Go Programming Language*. Accessed: 2025-05-22. 2025. URL: <https://go.dev/>.
- [69] Hyperledger Foundation. *What is Hyperledger Fabric?* Accessed: 2025-05-22. 2025. URL: <https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html>.

- [70] Hyperledger Foundation. *Using CouchDB as the State Database*. Accessed: 2025-05-22. 2025. URL: https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_tutorial.html.
- [71] MongoDB Inc. *Why Use MongoDB?* Accessed: 2025-05-22. 2025. URL: https://www.mongodb.com/resources/products/fundamentals/why-use-mongodb?utm_source=chatgpt.com.
- [72] Postman Inc. *What is Postman?* Accessed: 2025-05-23. 2025. URL: <https://www.postman.com/product/what-is-postman/>.
- [73] Hyperledger Explorer Project. *Hyperledger Explorer Documentation: Introduction*. Accessed: 2025-05-26. 2024. URL: <https://blockchain-explorer.readthedocs.io/en/main/introduction.html>.
- [74] *Adapter for Brother PTouch Label Makers 8.2Ft Long Cord*. <https://www.amazon.com/dp/B079FXKH9>. Accessed: 2025-06-08. 2025.
- [75] *ATmega328P Microcontroller Board with Cable for Arduino Nano*. <https://www.amazon.com/dp/B00NLAMS9C>. Accessed: 2025-06-08. 2025.
- [76] Rees52. *DC 12V Electric Solenoid Lock Tongue Upward Assembly for Doors, Cabinet, Drawer*. Accessed: 2025-06-07. 2025. URL: <https://rees52.com/products/dc-12v-electric-solenoid-lock-tounge-upward-assembly-for-doors-cabinet-drawer-ab102>.
- [77] *RFID Card Duplicator 13.56MHz Encrypted Programmer*. <https://www.amazon.com/dp/B088FB92XL>. Accessed: 2025-06-08. 2025.

Appendix: Smart Contracts

```

1 package main
2
3 import (
4     "encoding/json" "fmt" "strconv" "time"
5
6     "github.com/hyperledger/fabric-chaincode-go/shim"
7     sc "github.com/hyperledger/fabric-protos-go/peer"
8 )
9
10
11 type ResidentsContract struct{}
12
13
14 func (rc *ResidentsContract) Init(stub
15     shim.ChaincodeStubInterface) sc.Response {
16     return shim.Success(nil)
17 }
18
19 func containsVisitorInfo(visitors []VisitorInfo, visitorId
20     string) bool {
21     for _, v := range visitors {
22         if v.VisitorId == visitorId {
23             return true
24         }
25     }
26     return false
27 }
28
29 func containsString(slice []string, value string) bool {
30     for _, v := range slice {
31         if v == value {
32             return true
33         }
34     }
35     return false
36 }
37
38 func generatePermanentQRCode(userId string) string {
39     return fmt.Sprintf("QR-RESIDENT-%s", userId)
40 }
41
42 func (rc *ResidentsContract) RegisterResident(stub
43     shim.ChaincodeStubInterface, args []string) sc.Response {
44     if len(args) < 8 {
45         return shim.Error(" Required: ResidentID, Name, Email,
46             Phone, Gender, MaritalStatus, ResidentType,
47             Apartment")
48     }
49 }

```

```

45     residentId := args[0]
46     name := args[1]
47     email := args[2]
48     phone := args[3]
49     gender := args[4]
50     maritalStatus := args[5]
51     residentType := args[6]
52     apartment := args[7]
53
54     existingResident, err := stub.GetState("RESIDENT_" +
55         residentId)
56     if err != nil {
57         return shim.Error(fmt.Sprintf(" Failed to check existing
58             resident: %s", err.Error()))
59     }
60     if existingResident != nil {
61         return shim.Error(fmt.Sprintf(" Resident %s already
62             exists", residentId))
63     }
64
65     qrCode := residentId
66     timestamp := time.Now().Unix()
67
68     resident := Resident{
69         DocType:      "resident", queries
70         ResidentID:    residentId,
71         UserID:        residentId,
72         Name:          name,
73         Email:         email,
74         Phone:         phone,
75         Gender:        gender,
76         Apartment:     apartment,
77         MaritalStatus: maritalStatus,
78         ResidentType:  residentType,
79         QRCodeData:    qrCode,
80         QRCodeImage:   residentId + ".png",
81         CreatedAt:     timestamp,
82         UpdatedAt:     timestamp,
83     }
84
85     residentBytes, err := json.Marshal(resident)
86     if err != nil {
87         return shim.Error(fmt.Sprintf(" Failed to marshal
88             resident: %s", err.Error()))
89     }
90
91     err = stub.PutState("RESIDENT_"+residentId, residentBytes)
92     if err != nil {

```

```

91         return shim.Error(fmt.Sprintf(" Failed to store
           resident: %s", err.Error()))
92     }
93
94     return shim.Success(residentBytes)
95 }
96 func (rc *ResidentsContract) UpdateResident(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
97     if len(args) < 8 {
98         return shim.Error(" Required: ResidentID, Name, Email,
           Phone, Gender, MaritalStatus, ResidentType,
           Apartment")
99     }
100
101     residentId := args[0]
102     name := args[1]
103     email := args[2]
104     phone := args[3]
105     gender := args[4]
106     maritalStatus := args[5]
107     residentType := args[6]
108     apartment := args[7]
109
110     residentKey := "RESIDENT_" + residentId
111     existingResidentBytes, err := stub.GetState(residentKey)
112     if err != nil {
113         return shim.Error(fmt.Sprintf(" Failed to read resident:
           %s", err.Error()))
114     }
115     if existingResidentBytes == nil {
116         return shim.Error(fmt.Sprintf(" Resident %s does not
           exist", residentId))
117     }
118
119     var existingResident Resident
120     err = json.Unmarshal(existingResidentBytes,
        &existingResident)
121     if err != nil {
122         return shim.Error(fmt.Sprintf(" Failed to unmarshal
           resident: %s", err.Error()))
123     }
124
125     if existingResident.Apartment != apartment {
126         buildingBytes, err := stub.GetState("BUILDING_CONFIG")
127         if err != nil {
128             return shim.Error(fmt.Sprintf(" Failed to read
               building config: %s", err.Error()))
129         }
130         if buildingBytes == nil {
131             return shim.Error(" Building configuration not
               found")

```

```

132     }
133
134     var building BuildingConfig
135     err = json.Unmarshal(buildingBytes, &building)
136     if err != nil {
137         return shim.Error(fmt.Sprintf(" Failed to unmarshal
138             building config: %s", err.Error()))
139     }
140
141     residentsCount, err :=
142         rc.getResidentsCountInApartment(stub, apartment)
143     if err != nil {
144         return shim.Error(fmt.Sprintf(" Failed to count
145             residents: %s", err.Error()))
146     }
147
148     if residentsCount >= building.ResidentsPerApartment {
149         return shim.Error(" Maximum number of residents for
150             this apartment reached")
151     }
152
153     existingResident.Name = name
154     existingResident.Email = email
155     existingResident.Phone = phone
156     existingResident.Gender = gender
157     existingResident.MaritalStatus = maritalStatus
158     existingResident.ResidentType = residentType
159     existingResident.Apartment = apartment
160     existingResident.UpdatedAt = time.Now().Unix()
161
162     updatedResidentBytes, err := json.Marshal(existingResident)
163     if err != nil {
164         return shim.Error(fmt.Sprintf(" Failed to marshal
165             updated resident: %s", err.Error()))
166     }
167
168     err = stub.PutState(residentKey, updatedResidentBytes)
169     if err != nil {
170         return shim.Error(fmt.Sprintf(" Failed to update
171             resident: %s", err.Error()))
172     }
173
174     return shim.Success(updatedResidentBytes)
175 }
176
177 func (rc *ResidentsContract) BlockResident(stub
178     shim.ChaincodeStubInterface, args []string) sc.Response {
179     if len(args) < 6 {

```

```

175         return shim.Error(" Required: ResidentID, Reason,
176             BlockedBy, FromDate, FromTime, ToDate, ToTime")
177     }
178
179     residentId := args[0]
180     reason := args[1]
181     blockedBy := args[2]
182     fromDate := args[3]
183     fromTime := args[4]
184     toDate := args[5]
185     toTime := args[6]
186
187     residentKey := "RESIDENT_" + residentId
188     residentBytes, err := stub.GetState(residentKey)
189     if err != nil {
190         return shim.Error(fmt.Sprintf(" Failed to read resident:
191             %s", err.Error()))
192     }
193     if residentBytes == nil {
194         return shim.Error(fmt.Sprintf(" Resident %s does not
195             exist", residentId))
196     }
197
198     blockKey := "BLOCK_" + residentId
199     existingBlock, _ := stub.GetState(blockKey)
200     if existingBlock != nil {
201         return shim.Error(fmt.Sprintf(" Resident %s is already
202             blocked", residentId))
203     }
204
205     block := BlockRecord{
206         DocType:      "block",
207         ResidentID:    residentId,
208         Reason:        reason,
209         BlockedBy:     blockedBy,
210         FromDateTime:  fmt.Sprintf("%sT%s", fromDate, fromTime),
211         ToDateTime:    fmt.Sprintf("%sT%s", toDate, toTime),
212         CreatedAt:     time.Now().Unix(),
213     }
214
215     blockBytes, err := json.Marshal(block)
216     if err != nil {
217         return shim.Error(fmt.Sprintf(" Failed to marshal block
218             record: %s", err.Error()))
219     }
220
221     err = stub.PutState(blockKey, blockBytes)
222     if err != nil {

```

```

220         return shim.Error(fmt.Sprintf(" Failed to block
           resident: %s", err.Error()))
221     }
222
223
224     var resident Resident
225     err = json.Unmarshal(residentBytes, &resident)
226     if err != nil {
227         return shim.Error(fmt.Sprintf(" Failed to unmarshal
           resident: %s", err.Error()))
228     }
229
230     resident.IsBlocked = true
231     resident.UpdatedAt = time.Now().Unix()
232
233     updatedResidentBytes, _ := json.Marshal(resident)
234     stub.PutState(residentKey, updatedResidentBytes)
235
236     return shim.Success(blockBytes)
237 }
238 func (rc *ResidentsContract) GetResident(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
239
240     if len(args) != 1 {
241         return shim.Error(" Required: ResidentID")
242     }
243
244     residentId := args[0]
245
246
247     residentBytes, err := stub.GetState("RESIDENT_" + residentId)
248     if err != nil {
249         return shim.Error(fmt.Sprintf(" Failed to read resident:
           %s", err.Error()))
250     }
251     if residentBytes == nil {
252         return shim.Error(fmt.Sprintf(" Resident %s does not
           exist", residentId))
253     }
254
255     return shim.Success(residentBytes)
256 }
257 func (rc *ResidentsContract) UnblockResident(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
258     if len(args) < 1 {
259         return shim.Error(" Required: ResidentID")
260     }
261
262     residentId := args[0]
263
264

```



```

265     residentKey := "RESIDENT_" + residentId
266     residentBytes, err := stub.GetState(residentKey)
267     if err != nil {
268         return shim.Error(fmt.Sprintf(" Failed to read resident:
269             %s", err.Error()))
270     }
271     if residentBytes == nil {
272         return shim.Error(fmt.Sprintf(" Resident %s does not
273             exist", residentId))
274     }
275     blockKey := "BLOCK_" + residentId
276     blockBytes, err := stub.GetState(blockKey)
277     if err != nil {
278         return shim.Error(fmt.Sprintf(" Failed to read block
279             record: %s", err.Error()))
280     }
281     if blockBytes == nil {
282         return shim.Error(fmt.Sprintf(" Resident %s is not
283             blocked", residentId))
284     }
285     err = stub.DelState(blockKey)
286     if err != nil {
287         return shim.Error(fmt.Sprintf(" Failed to unblock
288             resident: %s", err.Error()))
289     }
290     var resident Resident
291     err = json.Unmarshal(residentBytes, &resident)
292     if err != nil {
293         return shim.Error(fmt.Sprintf(" Failed to unmarshal
294             resident: %s", err.Error()))
295     }
296     resident.IsBlocked = false
297     resident.UpdatedAt = time.Now().Unix()
298     updatedResidentBytes, _ := json.Marshal(resident)
299     stub.PutState(residentKey, updatedResidentBytes)
300     return shim.Success(nil)
301 }
302 func (rc *ResidentsContract) getResidentsCountInApartment(stub
303     shim.ChaincodeStubInterface, apartment string) (int, error) {
304     queryString := fmt.Sprintf("{
305         \"selector\": {
306             \"docType\": \"resident\",
307             \"Apartment\": \"%s\"
308         }

```

```

309     }, apartment)
310
311     resultsIterator, err := stub.GetQueryResult(queryString)
312     if err != nil {
313         return 0, err
314     }
315     defer resultsIterator.Close()
316
317     count := 0
318     for resultsIterator.HasNext() {
319         _, err := resultsIterator.Next()
320         if err != nil {
321             return 0, err
322         }
323         count++
324     }
325
326     return count, nil
327 }
328
329
330 func generateVisitorQRCode(visitorId string) string {
331     currentTime := time.Now().Format("2006-01-02 15:04:05")
332     return fmt.Sprintf("QR-VISITOR-%s-%s", visitorId, currentTime)
333 }
334
335
336 func (rc *ResidentsContract) AddVisitor(stub
337     shim.ChaincodeStubInterface, args []string) sc.Response {
338     if len(args) < 7 {
339         return shim.Error(" Required: ResidentID, VisitorID,
340             FullName, Phone, VisitTimeFrom, VisitTimeTo,
341             Relationship")
342     }
343
344     residentId := args[0]
345     visitorId := args[1]
346     fullName := args[2]
347     phone := args[3]
348     visitTimeFrom := args[4]
349     visitTimeTo := args[5]
350     relationship := args[6]
351
352     residentKey := "RESIDENT_" + residentId
353     residentBytes, err := stub.GetState(residentKey)
354     if err != nil {
355         return shim.Error(fmt.Sprintf(" Failed to read resident:
356             %s", err.Error()))
357     }
358     if residentBytes == nil {

```

```

355         return shim.Error(fmt.Sprintf(" Resident %s not found",
356             residentId))
357     }
358     var resident Resident
359     err = json.Unmarshal(residentBytes, &resident)
360     if err != nil {
361         return shim.Error(fmt.Sprintf(" Failed to unmarshal
362             resident: %s", err.Error()))
363     }
364     for _, v := range resident.Visitors {
365         if v.VisitorId == visitorId {
366             return shim.Error(fmt.Sprintf(" Visitor %s already
367                 exists for resident %s", visitorId, residentId))
368         }
369     }
370     newVisitor := VisitorInfo{
371         VisitorId:      visitorId,
372         FullName:       fullName,
373         Phone:          phone,
374         VisitTimeFrom:  visitTimeFrom,
375         VisitTimeTo:    visitTimeTo,
376         Relationship:    relationship,
377         QRCodeData:     visitorId,
378         Status:         "Active",
379         CreatedAt:      time.Now().Unix(),
380     }
381     resident.Visitors = append(resident.Visitors, newVisitor)
382     resident.UpdatedAt = time.Now().Unix()
383
384     updatedResidentBytes, err := json.Marshal(resident)
385     if err != nil {
386         return shim.Error(fmt.Sprintf(" Failed to marshal
387             resident: %s", err.Error()))
388     }
389     err = stub.PutState(residentKey, updatedResidentBytes)
390     if err != nil {
391         return shim.Error(fmt.Sprintf(" Failed to update
392             resident: %s", err.Error()))
393     }
394
395     response := map[string]interface{}{
396         "success": true,
397         "visitor": newVisitor,
398     }
399     responseBytes, _ := json.Marshal(response)
400     return shim.Success(responseBytes)

```

```

401 func (rc *ResidentsContract) GetVisitors(stub
402 shim.ChaincodeStubInterface, args []string) sc.Response {
403     if len(args) < 1 {
404         return shim.Error(" ResidentID required")
405     }
406
407     residentId := args[0]
408     residentKey := "RESIDENT_" + residentId
409     residentBytes, err := stub.GetState(residentKey)
410     if err != nil {
411         return shim.Error(fmt.Sprintf(" Failed to read resident:
412             %s", err.Error()))
413     }
414     if residentBytes == nil {
415         return shim.Error(fmt.Sprintf(" Resident %s not found",
416             residentId))
417     }
418
419     var resident Resident
420     err = json.Unmarshal(residentBytes, &resident)
421     if err != nil {
422         return shim.Error(fmt.Sprintf(" Failed to unmarshal
423             resident: %s", err.Error()))
424     }
425
426     response := map[string]interface{}{
427         "success": true,
428         "visitors": resident.Visitors,
429     }
430     responseBytes, _ := json.Marshal(response)
431     return shim.Success(responseBytes)
432 }
433
434 func (rc *ResidentsContract) GetVisitor(stub
435 shim.ChaincodeStubInterface, args []string) sc.Response {
436     if len(args) < 2 {
437         return shim.Error(" ResidentID and VisitorID required")
438     }
439
440     residentId := args[0]
441     visitorId := args[1]
442
443     residentKey := "RESIDENT_" + residentId
444     residentBytes, err := stub.GetState(residentKey)
445     if err != nil {
446         return shim.Error(fmt.Sprintf(" Failed to read resident:
447             %s", err.Error()))
448     }
449     if residentBytes == nil {
450         return shim.Error(fmt.Sprintf(" Resident %s not found",
451             residentId))

```

```

445     }
446
447     var resident Resident
448     err = json.Unmarshal(residentBytes, &resident)
449     if err != nil {
450         return shim.Error(fmt.Sprintf(" Failed to unmarshal
451             resident: %s", err.Error()))
452     }
453     for _, visitor := range resident.Visitors {
454         if visitor.VisitorId == visitorId {
455             response := map[string]interface{}{
456                 "success": true,
457                 "visitor": visitor,
458             }
459             responseBytes, _ := json.Marshal(response)
460             return shim.Success(responseBytes)
461         }
462     }
463     return shim.Error(fmt.Sprintf(" Visitor %s not found for
464         resident %s", visitorId, residentId))
465 }
466 func (rc *ResidentsContract) BlockVisitor(stub
467     shim.ChaincodeStubInterface, args []string) sc.Response {
468     if len(args) < 6 {
469         return shim.Error(" Required: VisitorID, ResidentID,
470             Reason, FromDate, FromTime, ToDate, ToTime,
471             BlockedBy")
472     }
473
474     visitorId := args[0]
475     residentId := args[1]
476     reason := args[2]
477     fromDate := args[3]
478     fromTime := args[4]
479     toDate := args[5]
480     toTime := args[6]
481     blockedBy := args[7]
482
483     fromDateTime, err := time.Parse("2006-01-02T15:04",
484         fmt.Sprintf("%sT%s", fromDate, fromTime))
485     if err != nil {
486         return shim.Error(fmt.Sprintf(" Invalid FromDateTime
487             format: %s", err.Error()))
488     }
489
490     toDateTime, err := time.Parse("2006-01-02T15:04",
491         fmt.Sprintf("%sT%s", toDate, toTime))
492     if err != nil {
493         return shim.Error(fmt.Sprintf(" Invalid ToDateTime
494             format: %s", err.Error()))

```

```

487     }
488
489     residentKey := "RESIDENT_" + residentId
490     residentBytes, err := stub.GetState(residentKey)
491     if err != nil {
492         return shim.Error(fmt.Sprintf(" Failed to read resident:
493             %s", err.Error()))
494     }
495     if residentBytes == nil {
496         return shim.Error(fmt.Sprintf(" Resident %s not found",
497             residentId))
498     }
499
500     var resident Resident
501     err = json.Unmarshal(residentBytes, &resident)
502     if err != nil {
503         return shim.Error(fmt.Sprintf(" Failed to unmarshal
504             resident: %s", err.Error()))
505     }
506
507     visitorFound := false
508     for i, visitor := range resident.Visitors {
509         if visitor.VisitorId == visitorId {
510             if visitor.Status == "Blocked" {
511                 return shim.Error(fmt.Sprintf(" Visitor %s is
512                     already blocked", visitorId))
513             }
514
515             blockId := fmt.Sprintf("BLOCK_%s_%d", visitorId,
516                 time.Now().Unix())
517             block := BlockInfo{
518                 BlockId:      blockId,
519                 VisitorId:    visitorId,
520                 Reason:       reason,
521                 BlockedBy:    blockedBy,
522                 FromDateTime: fromDateTime.Unix(),
523                 ToDateTime:    toDateTime.Unix(),
524                 CreatedAt:    time.Now().Unix(),
525             }
526
527             resident.Visitors[i].Status = "Blocked"
528             resident.Visitors[i].CurrentBlock = blockId
529
530             blockBytes, err := json.Marshal(block)
531             if err != nil {
532                 return shim.Error(fmt.Sprintf(" Failed to
533                     marshal block: %s", err.Error()))

```

```

532     }
533     err = stub.PutState(blockId, blockBytes)
534     if err != nil {
535         return shim.Error(fmt.Sprintf(" Failed to save
536             block: %s", err.Error()))
537     }
538     visitorFound = true
539     break
540 }
541 }
542
543 if !visitorFound {
544     return shim.Error(fmt.Sprintf(" Visitor %s not found for
545         resident %s", visitorId, residentId))
546 }
547
548 updatedResidentBytes, err := json.Marshal(resident)
549 if err != nil {
550     return shim.Error(fmt.Sprintf(" Failed to marshal
551         resident: %s", err.Error()))
552 }
553
554 err = stub.PutState(residentKey, updatedResidentBytes)
555 if err != nil {
556     return shim.Error(fmt.Sprintf("Failed to update
557         resident: %s", err.Error()))
558 }
559
560 return shim.Success([]byte(fmt.Sprintf(" Visitor %s blocked
561     successfully", visitorId)))
562 }
563
564 func (rc *ResidentsContract) UnblockVisitor(stub
565     shim.ChaincodeStubInterface, args []string) sc.Response {
566     if len(args) < 2 {
567         return shim.Error(" Required: VisitorID, ResidentID")
568     }
569
570     visitorId := args[0]
571     residentId := args[1]
572     residentKey := "RESIDENT_" + residentId
573     residentBytes, err := stub.GetState(residentKey)
574     if err != nil {
575         return shim.Error(fmt.Sprintf(" Failed to read resident:
576             %s", err.Error()))
577     }
578     if residentBytes == nil {
579         return shim.Error(fmt.Sprintf(" Resident %s not found",
580             residentId))

```

```

575     }
576
577     var resident Resident
578     err = json.Unmarshal(residentBytes, &resident)
579     if err != nil {
580         return shim.Error(fmt.Sprintf(" Failed to unmarshal
581             resident: %s", err.Error()))
582     }
583
584     visitorFound := false
585     for i, visitor := range resident.Visitors {
586         if visitor.VisitorId == visitorId {
587             if visitor.Status != "Blocked" {
588                 return shim.Error(fmt.Sprintf(" Visitor %s is
589                     not blocked", visitorId))
590             }
591
592             err = stub.DelState(visitor.CurrentBlock)
593             if err != nil {
594                 return shim.Error(fmt.Sprintf(" Failed to delete
595                     block record: %s", err.Error()))
596             }
597
598             resident.Visitors[i].Status = "Active"
599             resident.Visitors[i].CurrentBlock = ""
600
601             visitorFound = true
602             break
603         }
604     }
605
606     if !visitorFound {
607         return shim.Error(fmt.Sprintf(" Visitor %s not found for
608             resident %s", visitorId, residentId))
609     }
610
611     updatedResidentBytes, err := json.Marshal(resident)
612     if err != nil {
613         return shim.Error(fmt.Sprintf(" Failed to marshal
614             resident: %s", err.Error()))
615     }
616
617     err = stub.PutState(residentKey, updatedResidentBytes)
618     if err != nil {
619         return shim.Error(fmt.Sprintf(" Failed to update
620             resident: %s", err.Error()))
621     }
622
623     return shim.Success([]byte(fmt.Sprintf(" Visitor %s
624         unblocked successfully", visitorId)))

```



```

619 }
620
621
622 func (rc *ResidentsContract) UpdateVisitor(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
623     if len(args) < 4 {
624         return shim.Error(" Required: ResidentID, VisitorID,
Phone, VisitTimeFrom, VisitTimeTo")
625     }
626
627     residentId := args[0]
628     visitorId := args[1]
629     phone := args[2]
630     visitTimeFrom := args[3]
631     visitTimeTo := args[4]
632
633     residentKey := "RESIDENT_" + residentId
634     residentBytes, err := stub.GetState(residentKey)
635     if err != nil {
636         return shim.Error(fmt.Sprintf(" Failed to read resident:
%s", err.Error()))
637     }
638     if residentBytes == nil {
639         return shim.Error(fmt.Sprintf(" Resident %s not found",
residentId))
640     }
641
642     var resident Resident
643     err = json.Unmarshal(residentBytes, &resident)
644     if err != nil {
645         return shim.Error(fmt.Sprintf(" Failed to unmarshal
resident: %s", err.Error()))
646     }
647
648     visitorFound := false
649     for i, visitor := range resident.Visitors {
650         if visitor.VisitorId == visitorId {
651             resident.Visitors[i].Phone = phone
652             resident.Visitors[i].VisitTimeFrom = visitTimeFrom
653             resident.Visitors[i].VisitTimeTo = visitTimeTo
654             resident.UpdatedAt = time.Now().Unix()
655             visitorFound = true
656             break
657         }
658     }
659
660     if !visitorFound {
661         return shim.Error(fmt.Sprintf(" Visitor %s not found for
resident %s", visitorId, residentId))
662     }
663

```

```

664     updatedResidentBytes, err := json.Marshal(resident)
665     if err != nil {
666         return shim.Error(fmt.Sprintf(" Failed to marshal
        resident: %s", err.Error()))
667     }
668
669     err = stub.PutState(residentKey, updatedResidentBytes)
670     if err != nil {
671         return shim.Error(fmt.Sprintf(" Failed to update
        resident: %s", err.Error()))
672     }
673
674     return shim.Success([]byte(fmt.Sprintf(" Visitor %s updated
        successfully", visitorId)))
675 }
676
677 func (rc *ResidentsContract) AddVisitRequest(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
678     if len(args) < 11 {
679         return shim.Error(" Required: RequestID, CreatedBy,
        TargetResident, VisitorName, VisitorPhone, Type,
        VisitPurpose, CustomReason, VisitTimeFrom,
        VisitTimeTo, VisitDate")
680     }
681     requestId := args[0]
682     createdBy := args[1]
683     targetResident := args[2]
684     visitorName := args[3]
685     visitorPhone := args[4]
686     visitType := args[5]
687     visitPurpose := args[6]
688     customReason := args[7]
689     visitTimeFrom := args[8]
690     visitTimeTo := args[9]
691     visitDate := args[10]
692
693     newRequest := VisitRequest{
694         RequestID:      requestId,
695         CreatedBy:       createdBy,
696         TargetResident:  targetResident,
697         VisitorName:     visitorName,
698         VisitorPhone:    visitorPhone,
699         Type:            visitType,
700         VisitPurpose:    visitPurpose,
701         CustomReason:    customReason,
702         VisitTimeFrom:   visitTimeFrom,
703         VisitTimeTo:     visitTimeTo,
704         VisitDate:       visitDate,
705         Status:          "Pending",
706         CreatedAt:       time.Now().Unix(),
707     }

```

```

708 requestBytes, err := json.Marshal(newRequest)
709 if err != nil {
710     return shim.Error(fmt.Sprintf(" Failed to marshal visit
       request: %s", err.Error()))
711 }
712
713 requestKey := fmt.Sprintf("VISITREQUEST_%s", requestId)
714 err = stub.PutState(requestKey, requestBytes)
715 if err != nil {
716     return shim.Error(fmt.Sprintf(" Failed to store visit
       request: %s", err.Error()))
717 }
718
719 return shim.Success(requestBytes)
720 }
721
722 func (rc *ResidentsContract) UpdateVisitRequestStatus(stub
shim.ChaincodeStubInterface, args []string) sc.Response {
723     if len(args) < 3 {
724         return shim.Error(" Required arguments: RequestID,
       Status, RequestedBy")
725     }
726
727     requestID := args[0]
728     status := args[1]
729     requestedBy := args[2]
730
731     allowedStatuses := map[string]bool{
732         "accepted": true,
733         "rejected": true,
734     }
735
736     if !allowedStatuses[status] {
737         return shim.Error(" Invalid status value.")
738     }
739
740     requestKey := fmt.Sprintf("VISITREQUEST_%s", requestID)
741
742     requestBytes, err := stub.GetState(requestKey)
743     if err != nil {
744         return shim.Error(fmt.Sprintf(" Failed to get visit
       request: %s", err.Error()))
745     }
746     if requestBytes == nil {
747         return shim.Error(" Visit request not found.")
748     }
749
750     var request VisitRequest
751     err = json.Unmarshal(requestBytes, &request)
752     if err != nil {

```

```

753         return shim.Error(fmt.Sprintf(" Failed to parse visit
754             request: %s", err.Error()))
755     }
756     request.Status = status
757     request.UpdatedAt = time.Now().Unix()
758     request.StatusChangedBy = requestedBy
759
760     updatedBytes, err := json.Marshal(request)
761     if err != nil {
762         return shim.Error(fmt.Sprintf(" Failed to marshal
763             updated visit request: %s", err.Error()))
764     }
765     err = stub.PutState(requestKey, updatedBytes)
766     if err != nil {
767         return shim.Error(fmt.Sprintf(" Failed to store updated
768             request: %s", err.Error()))
769     }
770     return shim.Success(updatedBytes)
771 }
772
773
774 func (rc *ResidentsContract) GetAllResidents(stub
775     shim.ChaincodeStubInterface, args []string) sc.Response {
776     query := `{
777         "selector": {
778             "DocType": "resident"
779         }
780     }`
781
782     resultsIterator, err := stub.GetQueryResult(query)
783     if err != nil {
784         return shim.Error(fmt.Sprintf(" Query failed: %s",
785             err.Error()))
786     }
787     defer resultsIterator.Close()
788
789     var residents []Resident
790     for resultsIterator.HasNext() {
791         queryResponse, err := resultsIterator.Next()
792         if err != nil {
793             return shim.Error(fmt.Sprintf(" Failed to parse
794                 resident: %s", err.Error()))
795         }
796
797         var resident Resident
798         err = json.Unmarshal(queryResponse.Value, &resident)

```

```

798     if err != nil {
799         return shim.Error(fmt.Sprintf(" Failed to unmarshal
800             resident: %s", err.Error()))
801     }
802     residents = append(residents, resident)
803 }
804 residentsJSON, err := json.Marshal(residents)
805 if err != nil {
806     return shim.Error(fmt.Sprintf(" Failed to marshal
807         residents: %s", err.Error()))
808 }
809 return shim.Success(residentsJSON)
810 }
811
812 func (rc *ResidentsContract) SaveLogToChain(stub
813     shim.ChaincodeStubInterface, args []string) sc.Response {
814     if len(args) != 3 {
815         return shim.Error(" Required: RequestID, ActionType,
816             Timestamp")
817     }
818
819     logId := fmt.Sprintf("LOG_%s_%d", args[0],
820         time.Now().UnixNano())
821
822     log := struct {
823         LogID      string `json:"logId"`
824         RequestID  string `json:"requestId"`
825         Type       string `json:"type"`
826         Timestamp  int64  `json:"timestamp"`
827     }{
828         LogID:      logId,
829         RequestID:  args[0],
830         Type:       args[1],
831     }
832
833     parsedTime, err := strconv.ParseInt(args[2], 10, 64)
834     if err != nil {
835         return shim.Error(" Invalid timestamp format")
836     }
837     log.Timestamp = parsedTime
838
839     logBytes, err := json.Marshal(log)
840     if err != nil {
841         return shim.Error(fmt.Sprintf(" Failed to marshal log:
842             %s", err.Error()))
843     }
844
845     err = stub.PutState(logId, logBytes)
846     if err != nil {

```

```

843         return shim.Error(fmt.Sprintf(" Failed to save log to
            chain: %s", err.Error()))
844     }
845
846     return shim.Success([]byte(logId))
847 }
848 func (rc *ResidentsContract) GetLastLogByResident(stub
    shim.ChaincodeStubInterface, args []string) sc.Response {
849     if len(args) != 1 {
850         return shim.Error(" Required: ResidentID")
851     }
852     residentID := args[0]
853     queryString := fmt.Sprintf("{
854         "selector": {
855             "residentId": "%s"
856         },
857         "sort": [{"timestamp": "desc"}],
858         "limit": 1
859     }", residentID)
860
861     resultsIterator, err := stub.GetQueryResult(queryString)
862     if err != nil {
863         return shim.Error(fmt.Sprintf(" Failed to query logs:
            %s", err.Error()))
864     }
865     defer resultsIterator.Close()
866
867     if !resultsIterator.HasNext() {
868         return shim.Error(" No logs found for the given resident
            ID")
869     }
870
871     result, err := resultsIterator.Next()
872     if err != nil {
873         return shim.Error(fmt.Sprintf(" Failed to read query
            result: %s", err.Error()))
874     }
875     var logEntry struct {
876         Type string `json:"type"`
877     }
878     err = json.Unmarshal(result.Value, &logEntry)
879     if err != nil {
880         return shim.Error(fmt.Sprintf(" Failed to parse log
            entry: %s", err.Error()))
881     }
882
883     return shim.Success([]byte(logEntry.Type))
884 }
885 func (rc *ResidentsContract) GetVisitRequest(stub
    shim.ChaincodeStubInterface, args []string) sc.Response {
886     if len(args) < 1 {

```

```

887     return shim.Error(" Required: RequestID")
888 }
889
890 requestId := args[0]
891 requestKey := fmt.Sprintf("VISITREQUEST_%s", requestId)
892
893 requestBytes, err := stub.GetState(requestKey)
894 if err != nil {
895     return shim.Error(fmt.Sprintf(" Failed to read visit
896         request: %s", err.Error()))
897 }
898
899 if requestBytes == nil {
900     return shim.Error("Visit request not found")
901 }
902
903 return shim.Success(requestBytes)
904 }
905
906 func contains(slice []string, item string) bool {
907     for _, s := range slice {
908         if s == item {
909             return true
910         }
911     }
912     return false
913 }
914
915 func main() {
916     err := shim.Start(new(ResidentsContract))
917     if err != nil {
918         fmt.Printf(" Error starting ResidentsContract: %s", err)
919     }
920 }

```

Listing 1: Chaincode for Asset Transfer