# ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to Mr. Amith K S, Assistant Professor, who handled the Blockchain Technology subject, for introducing the practical aspects of smart contract development and guiding us through hands-on implementation. His approach to teaching encouraged a deeper understanding of blockchain fundamentals and their real-world applications. I extend my heartfelt thanks to Dr. Ashok Kumar T, Principal, and Mrs. Veena Bhat, Head of the Department of Artificial Intelligence and Data Science, for providing the necessary resources and academic environment that supported this project. I also acknowledge the Management of SDM Institute of Technology, Ujire, for offering excellent infrastructure, including access to laboratories and digital tools required for project development. Lastly, I sincerely appreciate the support and encouragement from all faculty members and non-teaching staff of the department who contributed directly or indirectly to the successful completion of this project.

Siri H R

# ABSTRACT

The Smart Rental System, known as RentSmart, is a decentralized application (dApp) built on the Ethereum blockchain that aims to simplify and automate the process of room rentals. Traditional rental systems often involve manual contracts, delays, and reliance on intermediaries, which can lead to disputes and inefficiencies. RentSmart addresses these challenges by utilizing smart contracts to manage rental agreements, payments, and lease terminations in a transparent and automated manner.In this system, landlords can list rooms by specifying essential details such as location, rent per minute, and security deposit. Tenants can browse available rooms and initiate a rental by signing a digital lease agreement directly through the smart contract. The agreement defines the duration of the lease and facilitates immediate payment of rent and security deposits, ensuring a trustless transaction between both parties. Rent is calculated and paid in real-time, allowing for short-term and flexible usage models. The contract also records timestamps for each payment and automatically updates the lease status.Tenants can continue paying rent based on time usage and may end the lease when needed. Once the lease duration ends, or upon early termination, the contract can be programmed to release deposits or settle final payments accordingly. This approach not only enhances operational efficiency but also ensures data immutability and security, as all transactions are stored on the blockchain.

Overall, RentSmart offers a modern, decentralized alternative to traditional rental systems, reducing administrative overhead while increasing fairness and transparency for both landlords and tenants.

# Tables of Contents

# List of Figures



# List of Tables

# Chapter 1

# Introduction

The RentSmart smart contract is a decentralized rental management system built on the Ethereum blockchain using Solidity. It streamlines the process of listing, leasing, and managing rental rooms by automating interactions between landlords and tenants. Designed to operate on a per-minute rent model, the contract enables landlords to list rooms with specified rent rates and security deposits, while allowing tenants to lease rooms by paying the required amount upfront. Once an agreement is signed, tenants can continue to pay rent at fixed intervals, and landlords receive payments directly through the contract. The system ensures transparency and trust by recording agreements on-chain, enforcing time-based rent payments, and managing lease activity. Key features include real-time occupancy tracking, payment logging, and lease duration management. Access control is implemented through modifiers to restrict functions to landlords or tenants. RentSmart promotes a fair, efficient, and automated rental experience without intermediaries.

## 1.1 Overview of the Project

The Smart Rental System aims to revolutionize the traditional rental process by integrating blockchain technology to ensure transparency, security, and automation. In conventional rental agreements, issues like disputes, delayed payments, lack of trust, and manipulation of contracts often arise. This project addresses these challenges by leveraging smart contracts on the Ethereum blockchain to automate and enforce rental agreements without the need for intermediaries. The system enables landlords to list rooms with specified rent, location, and security deposit. Tenants can sign rental agreements by paying the rent and deposit through a secure blockchain transaction. The smart contract automatically manages lease durations, enforces payment terms, and logs every transaction immutably.

Key features include:

- **Room Listing**: Landlords can publish vacant rooms with relevant details.
- **Smart Agreement Signing**: Tenants initiate rental agreements by sending the required payment.

- **Automated Rent Collection**: Rent payments are managed through time-based logic and require tenant actions.
- **Lease Tracking**: The contract records agreement status and ensures tenants cannot exploit lease terms.
- **Transparency & Security**: All actions are recorded on-chain, offering trust and auditability for both parties.

This decentralized approach provides a **trustless**, **automated**, and **tamper-proof rental ecosystem** that significantly improves efficiency and reduces administrative overhead.

## 1.2 Problem Statement

The traditional rental system is plagued by inefficiencies, trust deficits, and manual procedures that frequently result in disputes between landlords and tenants. Common issues include forged rental agreements, delayed or missed rent payments, lack of transparency in terms and conditions, and concerns over the security of financial transactions. Additionally, the reliance on third-party intermediaries not only complicates the process but also increases operational costs and delays. In centralized systems, both landlords and tenants face limited protection in cases of fraud or non-compliance, often requiring costly legal intervention to enforce agreements. The absence of automation further exacerbates the problem, making it difficult to manage rental cycles, track payments, and enforce lease terms efficiently.

To address these challenges, there is a growing need for a decentralized, transparent, and automated rental system that ensures secure and verifiable rental agreements, eliminates the need for intermediaries, reduces costs and delays, automates rent processing and lease enforcement, and builds trust through the use of immutable blockchain records. This project leverages blockchain-based smart contracts to offer a tamper-proof and self-executing rental platform, significantly enhancing security, transparency, and efficiency in rental management.

## 1.3 Objectives of the Project

The primary objective of the Smart Rental System is to develop a decentralized rental management platform that leverages blockchain technology to automate and secure the process of renting properties. The specific goals of this project include:

- **To eliminate the reliance on third-party intermediaries** by using smart contracts to directly manage agreements between landlords and tenants.
- **To provide a secure and transparent environment** where all transactions, agreements, and payments are recorded immutably on the blockchain.
- **To design and implement smart contracts** that handle room listings, agreement signing, rent payments, and lease management automatically.
- **To ensure trust and fairness** by enforcing contract terms through self-executing logic without the need for manual oversight or legal enforcement.

By achieving these objectives, the system aims to create a secure, cost-effective, and efficient rental process that benefits both landlords and tenants.

## 1.4 Scope of the Project

The **Smart Rental System** is a blockchain-based solution aimed at revolutionizing the traditional property rental process by making it more secure, transparent, and automated. This system utilizes the Ethereum blockchain to create a decentralized platform where landlords can list properties and tenants can securely sign lease agreements and make rent payments. By leveraging the power of smart contracts, the platform eliminates the need for intermediaries such as real estate agents or legal personnel in managing rental transactions, thus reducing both cost and complexity for all parties involved.

At the core of the system is a set of smart contracts that automate key rental functions. Landlords can use these contracts to list available rental properties along with terms such as rental amount, lease duration, and security deposit. Tenants, on the other hand, can view listings, initiate rental agreements, and make payments directly through the blockchain. Once a lease agreement is accepted by both parties, the smart contract becomes active and enforces the agreed-upon terms. Rent payments are scheduled and processed automatically based on predefined time intervals. This automation ensures timely payments and reduces the risk of defaults or late fees.

All financial transactions and rental records are stored securely and immutably on the Ethereum blockchain, providing a tamper-proof audit trail. This level of transparency helps build trust between landlords and tenants, as all actions taken during the rental lifecycle are recorded and verifiable. Additionally, the system includes basic role-based access control, ensuring that only

authorized users (landlords or tenants) can perform certain actions, such as updating property details or signing a lease. This enhances the overall security and integrity of the platform.

The scope of the Smart Rental System is intentionally limited to backend smart contract functionality. It does not include a graphical user interface, advanced dispute resolution features, or integrations with off-chain services such as banks or government ID verification systems. The focus is on developing a functional and secure foundation for property rentals using blockchain technology, which can be extended or integrated with additional components in future iterations.

By focusing on decentralization, automation, and transparency, the Smart Rental System addresses many of the inefficiencies and trust issues present in traditional rental processes. It represents a significant step forward in using blockchain for real-world applications, particularly in the real estate sector, where secure and verifiable transactions are crucial.

## 1.5 Methodology

The methodology outlines the systematic approach used in designing and developing the RentSmart smart contract. It includes both the research process that informed the contract's design and the technical steps taken during its implementation. This section ensures a clear understanding of how the project was conceptualized and executed.

### 1.5.1 Research Methodology

The development of the RentSmart smart contract was grounded in a structured research methodology to ensure effectiveness, security, and innovation. A blend of research techniques was employed to address the limitations of traditional rental systems. The process emphasized the use of blockchain for enhanced trust, automation, and transparency.

**Research Methodology Highlights:**

- **Exploratory Research:** Investigated issues in traditional rental systems like lack of transparency, delayed payments, and reliance on intermediaries.
- **Comparative Analysis:** Studied decentralized rental platforms and smart contract applications to identify design patterns, limitations, and enhancement opportunities.
- **Requirements Gathering:** Focused on decentralization, trustlessness, real-time payments, and secure lease agreements.

- **Platform & Language Selection:** Chose Ethereum for its maturity and Solidity for its compatibility and widespread adoption.
- **Security Considerations:** Reviewed smart contract vulnerabilities and security best practices to guide the implementation.

## 1.5.2 Development Methodology

The development of the **RentSmart** smart contract followed a structured and research-informed methodology aimed at addressing key challenges in the traditional rental ecosystem. The goal was to build a decentralized, secure, and transparent rental system that leverages blockchain's strengths to manage lease agreements, automate rent payments, and minimize the need for intermediaries. The methodology encompassed several phases: problem identification, requirement gathering, platform selection, contract design, implementation, and security validation.

**1. Problem Identification and Exploratory Research**

The initial phase involved conducting exploratory research to analyze issues in conventional rental systems. Challenges identified included delayed payments, disputes over lease terms, lack of transparency, and dependence on third parties such as agents or legal advisors. The research concluded that blockchain technology could help overcome these problems by providing automation, immutability, and trustless interactions.

**2. Requirement Gathering and Functional Specification**

Following the research, specific system requirements were defined to guide the design and development process. The core functional requirements included:

- Landlords should be able to list available rental rooms.
- Tenants should be able to sign rental agreements directly.
- Rent should be paid in real-time based on a time unit (e.g., per minute).
- Lease agreements should be automatically enforced and terminated after the specified duration.

- Deposits should be securely handled and traceable.
- The contract should be role-aware, distinguishing landlords from tenants.

### 3. Platform and Technology Selection

Ethereum was chosen as the blockchain platform due to its maturity, widespread adoption, and support for smart contract development. Its strong community and developer tools made it ideal for a project requiring reliable execution of automated agreements. Solidity, Ethereum's native contract language, was selected due to its compatibility with the Ethereum Virtual Machine (EVM) and comprehensive documentation.

### 4. Contract Architecture and Design

The smart contract architecture was designed to be modular, scalable, and secure. Two main data structures were defined:

- Room: Contains data about the rental unit, including landlord, rent per minute, deposit, location, availability, and tenant information.
- Agreement: Manages the lease, including lease duration, tenant, timestamps, and active status.

Mappings were used to efficiently store and retrieve room and agreement data. Events such as RoomListed, AgreementSigned, and RentPaid were implemented to allow for off-chain tracking and enhance transparency.

Access control was enforced using custom modifiers such as onlyLandlord, onlyTenant, and isLeaseActive, ensuring that only authorized users could perform sensitive operations.

### 5. Implementation and Security Practices

The contract implementation focused on correctness and security. Key functions included:

- listRoom: Allows landlords to register a room.
- signAgreement: Lets tenants initiate a lease and transfer deposit and rent.
- payRent: Enables tenants to make periodic rent payments based on elapsed time.
- getCurrentTime: Helps synchronize operations using the current block timestamp.

Security best practices were followed throughout, including avoiding re-entrancy vulnerabilities, validating input payments, and using Solidity 0.8.x features to prevent overflows. State updates were always made before external transfers to avoid attack vectors.

This structured methodology ensured that RentSmart was not only functional but also secure, transparent, and aligned with blockchain's core strengths.

# Chapter 2

# Literature Survey

Blockchain technology, particularly smart contracts on Ethereum, is revolutionizing the rental and lease management sector by eliminating intermediaries and enabling automated, trustless agreements. Traditional rental systems face challenges such as delayed payments, lack of transparency, and manual verification, which smart contracts can address through secure, decentralized automation.

Recent literature explores how Ethereum-based smart contracts can streamline property rentals by automating lease agreements, handling deposits, and enabling real-time rent payments. Solidity, the primary language for Ethereum smart contracts, offers tools such as mappings, structs, and modifiers that facilitate efficient and secure contract development.

A key focus in modern smart contract design is role-based access control, ensuring only landlords and tenants can perform specific actions. Additionally, the use of block.timestamp allows accurate enforcement of lease durations and rent cycles. Smart contract security remains a vital area of research, with studies addressing vulnerabilities like re-entrancy, access mismanagement, and logic errors through rigorous coding standards and auditing practices.

This literature survey highlights the evolution of decentralized rental systems and the application of blockchain to improve efficiency, transparency, and security. These insights have directly influenced the design of the RentSmart smart contract, which aims to modernize and automate the rental process in a secure and user-friendly way.

## 2.1 Introduction to Literature Survey

The literature survey for the development of the **RentSmart** smart contract focused on exploring existing research, tools, and decentralized applications relevant to blockchain-based rental systems. The primary objective was to gain a comprehensive understanding of how smart contracts and decentralized technologies have been used to address challenges in the property rental domain, such as lack of transparency, manual processes, delayed payments, and intermediary dependence.

Through this survey, various smart contract platforms, decentralized rental solutions, and Ethereum-based lease management applications were analyzed. Emphasis was placed on

understanding the design patterns, limitations, and security concerns associated with smart contracts. The review also covered best practices for smart contract development in Solidity, the implementation of secure payment flows, and the use of time-based logic for automated rent transactions.

This literature review served as a foundation for identifying the functional and technical gaps in current systems and helped inform the architecture and development of the RentSmart contract. It also provided insights into how blockchain can be effectively utilized to create trustless, automated, and tamper-proof rental agreements.

## 2.2 Existing Systems or Technologies

A study by Qingshui Xue et al. [1] presents a blockchain-based housing rental system designed to tackle issues like fraudulent listings, arbitrary fees, and lack of transparency in the rental market. The proposed solution employs an alliance blockchain network, enabling landlords and tenants to interact via smart contracts for secure lease agreements and automated rent payments. The system features dedicated modules for registration, listings, leasing, and smart contracts, ensuring data integrity and tamper-proof transactions. Experimental validation demonstrates enhanced security, transparency, and cost-efficiency, making the approach suitable for modernizing housing rental operations and supporting government oversight.

A study by Chen Qi-Long et al. [2] proposes a decentralized housing rental system leveraging blockchain and IPFS technologies to eliminate intermediaries and enhance transaction transparency. The system enables peer-to-peer sharing of rental listings, converting traditional lease agreements into Ethereum-based smart contracts for automated execution and legal assurance. Listing data is securely stored using the InterPlanetary File System (IPFS), ensuring tamper resistance and persistent access. The proposed solution enhances operational efficiency, data integrity, and legal protection, making it a viable model for secure and transparent rental ecosystems.

A study by André S. Proença et al. [3] introduces a GDPR-compliant blockchain platform designed to streamline residential rental processes by removing intermediaries such as notaries and real estate agents. The system utilizes smart contracts on a permissioned blockchain to enable secure, transparent, and tamper-proof management of rental agreements and payments. By addressing inefficiencies, high costs, data security, and lack of transparency in traditional rental systems, the proposed solution offers a legally sound and privacy-conscious approach to modernizing residential rentals.

A study by Andreas Bogner et al. [4] presents a decentralized application (DApp) prototype that leverages Ethereum smart contracts to enable peer-to-peer item sharing without intermediaries. The system allows users to list and rent items directly, with smart contracts automating transactions and enforcing agreements. Blockchain's immutability ensures transparent and trustless interactions, enhancing security and reducing reliance on centralized authorities. The proposed approach showcases the potential of decentralized sharing economies, delivering improved efficiency, user autonomy, and transactional integrity.

## 2.3 Gaps in Existing Solution

While blockchain-based systems for housing rentals and peer-to-peer sharing show promise in enhancing transparency, automation, and security, several limitations remain unaddressed. Most existing solutions focus on technical feasibility but often overlook practical deployment challenges, regulatory compliance, and user accessibility. The following gaps highlight key areas for further improvement and research:

1. **Lack of Cross-Platform Interoperability:** Existing solutions are built on isolated platforms (e.g., Ethereum or permissioned blockchains), making integration with other systems or platforms difficult.

2. **Scalability Limitations:** Public blockchains like Ethereum face scalability issues, including high transaction fees and delays during network congestion.

3. **Regulatory and Legal Ambiguity:** There is minimal integration with national or international legal frameworks, making enforcement of smart contracts in legal disputes unclear.

4. **User Experience and Accessibility:** Many systems lack user-friendly interfaces and require technical knowledge, which can hinder adoption by non-technical users.

5. **Limited Privacy Controls:** Blockchain's immutability conflicts with data privacy laws like GDPR, especially in systems that store user or contract data on-chain.

6. **Insufficient Off-Chain Integration:** Important real-world data such as identity verification, inspections, or maintenance histories are often poorly integrated or handled manually.

7. **Smart Contract Flexibility and Upgradability:** Many systems use rigid smart contracts that are difficult to update or customize once deployed, limiting adaptability to changing needs or regulations.

8. **Lack of Dispute Resolution Mechanisms:** There is little support for conflict resolution or arbitration if disagreements occur between parties in a smart contract.

9. **High Initial Setup and Maintenance Costs:** Deploying and maintaining blockchain infrastructure, especially for permissioned networks, can be resource-intensive.

10. **Low Stakeholder Collaboration:** Many solutions focus solely on tenants and landlords, with limited involvement from regulatory bodies, insurers, or financial institutions.

## 2.4 Summary of Literature Survey

The reviewed literature explores various blockchain-based solutions aimed at transforming traditional rental and sharing systems through decentralization, smart contracts, and secure data storage.

Several studies propose platforms that utilize Ethereum-based smart contracts to automate rental agreements and transactions, removing intermediaries such as real estate agents or notaries. These systems enhance transparency, reduce costs, and ensure tamper-proof records. For instance, Chen Qi-Long et al. and Andreas Bogner et al. develop decentralized applications (DApps) for peer-to-peer rentals, focusing on trust and security in user interactions.

Other works, such as those by Qingshui Xue et al. and André S. Proença et al., propose permissioned blockchain models for more controlled environments. These models emphasize GDPR compliance, government oversight, and legal protection, highlighting blockchain's potential in regulated sectors.

Some systems incorporate InterPlanetary File System (IPFS) for decentralized file storage, ensuring integrity and persistence of rental listings and documents.

Despite their innovations, the surveyed systems face common challenges, including limited scalability, legal ambiguity, poor user accessibility, and insufficient handling of off-chain data.

# Chapter 3

# Software and Hardware Requirements

## 3.1 Software Requirements

To develop RentSmart, a decentralized rental management system, a range of specialized software tools and technologies is required. These span across blockchain development, smart contract programming, user interface integration, and secure transaction handling. The platform relies on Ethereum for deployment, Solidity for contract logic, and Web3 technologies for frontend connectivity. Together, these components enable a trustless, automated rental system that streamlines lease agreements, rent payments, and tenant-landlord interactions in a secure and transparent manner.

### 3.1.1 Operating System

The RentSmart system is compatible with major operating systems including Windows, macOS, and Linux. It can be developed and deployed on any operating system that supports Ethereum development tools and Solidity compilers. This cross-platform compatibility ensures that developers can work in their preferred environments while maintaining full functionality and support for smart contract development and blockchain integration.

### 3.1.2 Tools and Frameworks

1. **Solidity Compiler (solc)**: To compile the smart contract code.
2. **Remix IDE** (online or desktop) or **Visual Studio Code** with Solidity extensions for development.
3. **Truffle / Hardhat** framework for deploying, testing, and managing smart contracts.
4. **Ganache** or other local Ethereum blockchain simulators for testing.
5. **Node.js** and **npm** for managing dependencies and running scripts.
6. **MetaMask** or similar Ethereum wallet for interacting with deployed contracts.

### 3.1.3 Libraries and APIs

1. **OpenZeppelin** (optional) for secure contract components and utilities.
2. **Web3.js** or **Ethers.js** libraries for front-end interaction with the Ethereum blockchain.
3. **Ethereum JSON-RPC API** for blockchain node communication.

## 3.2 Hardware Requirements

Due to the inclusion of blockchain operations such as smart contract compilation, deployment, and real-time transaction processing, the RentSmart project requires certain minimum hardware capabilities. These specifications are essential to ensure efficient contract development, smooth interaction with the Ethereum network, and reliable performance during both testing and live deployments. Adequate hardware resources also support the use of development tools like Remix IDE, Truffle, and local blockchain environments, ensuring a seamless experience throughout the development lifecycle.

### 3.2.1 Processor Specifications

1. **Recommended:** Quad-core processor or higher (e.g., Intel Core i5/i7, AMD Ryzen 5/7).
2. **Reason:** Running local Ethereum nodes (like Ganache), compiling smart contracts, running tests, and simultaneous development tasks benefit from multiple cores and higher clock speeds for faster execution.
3. **Minimum:** Dual-core processor (e.g., Intel Core i3 or equivalent).

### 3.2.2 RAM and Storage Requirements

1. **RAM:**
    a. **Recommended:** 16 GB or more.
    b. **Minimum:** 8 GB.
    c. **Reason:** Running blockchain simulators, development environments (IDEs), browsers with blockchain wallets (MetaMask), and test scripts simultaneously can be RAM-intensive.
2. **Storage:**
    a. **Recommended:** 100 GB SSD.

b. **Minimum:** 20 GB SSD/HDD.
c. **Reason:** SSD storage improves the speed of blockchain data access, compilation, and overall system responsiveness. The blockchain data, node logs, and project files accumulate over time.
d. Ensure sufficient free disk space for:
    i. Ethereum blockchain data (if running a full or archive node).
    ii. Project files, dependencies, and logs.

### 3.2.3 Other Hardware Needs

1. **Network Interface:**

    a. Stable and reasonably fast internet connection (minimum 10 Mbps recommended).

    b. Required for interacting with public Ethereum networks (testnets/mainnet) or remote blockchain nodes.

2. **Display:**

    a. Full HD (1920x1080) resolution or higher for comfortable coding and multi-window management.

    b. **Power Supply:**

    c. Reliable power source or UPS (Uninterruptible Power Supply) if running long tests or local nodes that need uninterrupted uptime.

3. **Optional:**

    a. **Hardware Wallet** (e.g., Ledger Nano S/X, Trezor) for secure transaction signing during deployment and interaction with smart contracts.

    b. **Dedicated development machine or virtual machine** if isolating blockchain development environment from general use.

# Chapter 4

# System Design

**RentSmart** is a decentralized platform developed to manage interactions between landlords and tenants. The system includes components like Ethereum smart contracts, wallet-based authentication, automated lease agreements, and on-chain rent payments, all integrated into a unified blockchain application.

## 4.1 Architecture Design

The system architecture of **RentSmart** illustrates the interaction between various components of the decentralized rental platform. It outlines how users (landlords and tenants) interact with the frontend DApp, which communicates with Ethereum smart contracts via blockchain nodes. The architecture emphasizes secure wallet-based authentication, smart contract-driven logic for lease and rent management, and real-time blockchain interaction for transparency and automation.

**Figure 4.1**

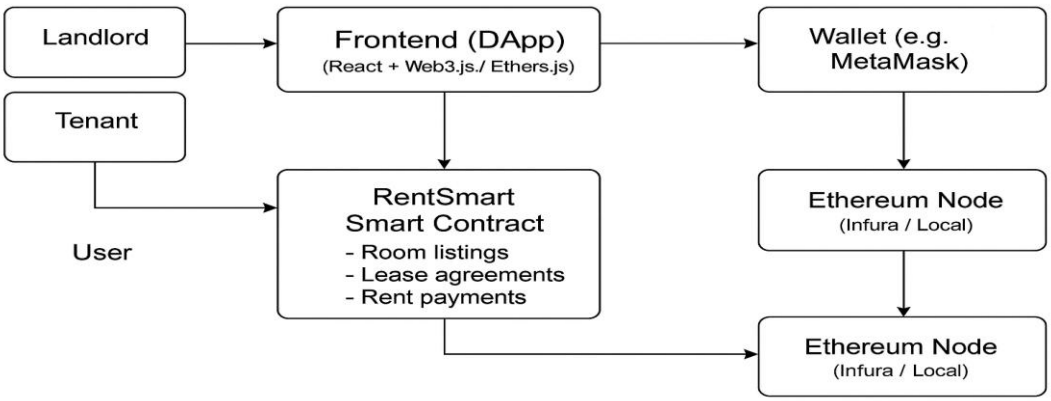


**Figure 4.1Architecture Design**

**Figure 4.1** shows the architectural diagram of **RentSmart**, which includes several key components:

1. **Frontend (DApp):**
    a. Built using **React** along with **Web3.js** or **Ethers.js**.
    b. Acts as the user interface for both landlords and tenants.

c. Facilitates interactions such as room listings, lease agreements, and rent payments.

2. **Wallet (e.g., MetaMask):**
    a. Used to authenticate and authorize transactions on behalf of users.
    b. Ensures secure signing and submission of blockchain transactions.
3. **Ethereum Node (Infura / Local):**
    a. Handles communication with the Ethereum blockchain.
    b. Processes and validates transactions, enabling interaction with the smart contract.
4. **RentSmart Smart Contract:**
    a. Core of the application deployed on the Ethereum blockchain.
    b. Manages:
        i. Room listings
        ii. Lease agreements
        iii. Rent payments
5. **Users (Landlord and Tenant):**
    a. Interact with the system via the frontend.
    b. Initiate and complete rental activities in a decentralized manner.

## 4.2 Module Description:

The Smart Rental System (RentSmart) is composed of multiple integrated modules, each responsible for fulfilling specific functions such as room management, lease handling, tenant-landlord interaction, and secure rent payments. These modules work in unison to provide a decentralized, transparent, and automated rental experience for both landlords and tenants. The system is built on Ethereum blockchain and ensures trustless interactions with robust security protocols and a scalable architecture.

### 4.2.1 User and Access Management

**T**his module manages access roles and identities for both landlords and tenants. Users interact with the platform via Web3-compatible wallets such as MetaMask, which provide cryptographic authentication using Ethereum addresses. Role-based interactions are enforced

using Solidity modifiers like onlyLandlord and onlyTenant, ensuring that only authorized users can perform sensitive actions such as rent collection or payment.

### 4.2.2 Room Listing and Management

Landlords can list their rental rooms using this module by providing room details such as name, location, rent per minute, and security deposit. Each room is uniquely identified and stored in a Room struct, and its availability status is tracked. An event RoomListed is emitted on successful listing, making this data traceable on-chain.

### 4.2.3 Lease Agreement Handling

This module enables tenants to sign lease agreements for vacant rooms. On initiating a lease, the tenant pays the first rent and a security deposit. A new Agreement struct is created to capture lease start time, duration, and tenant address. The lease is marked active and associated with the room. The AgreementSigned event ensures transparency and traceability of contract creation.

### 4.2.4 Rent Payment System

Tenants pay rent periodically through this module, with validations in place to ensure that the lease is active and rent is due. The system verifies if the time since the last payment has exceeded the minimum duration. Rent is transferred to the landlord's wallet, and a RentPaid event is triggered to confirm the transaction. This fully on-chain logic prevents payment fraud and ensures timely settlements.

### 4.2.5 Blockchain Interaction and Wallet Integration

All smart contract interactions require users to connect their wallets (e.g., MetaMask). This module ensures secure signing and submission of transactions to the Ethereum blockchain using libraries like Web3.js or Ethers.js in the frontend. The system supports interaction through Ethereum nodes via Infura or a local setup for testing and scalability.

### 4.2.6 Smart Contract State Monitoring

This module provides utility functions like getCurrentTime() to fetch blockchain timestamps. It helps in evaluating rent due times, lease expiry, and rent eligibility logic. The use of public

mappings allows frontend components to fetch real-time data directly from the contract for display and decision-making.

### 4.2.7 Event Logging and Transparency

The system emits structured events (RoomListed, AgreementSigned, RentPaid, LeaseEnded) to create a tamper-proof audit trail of all critical transactions. This enhances transparency, simplifies integration with third-party services, and supports real-time monitoring of rental activities via DApp frontends.

### 4.2.8 Security and Data Integrity

Security is enforced using Solidity access control and on-chain validations. Wallet-based authentication eliminates the need for traditional passwords. All financial transactions are handled in a trustless, decentralized manner. Since all interactions occur on the Ethereum blockchain, the integrity and immutability of rental records are inherently guaranteed.

## 4.3 Data Flow Diagram:

A Data Flow Diagram (DFD) visually represents how data moves within a system. It shows the flow of information between processes, data stores, and external entities. DFDs help in understanding system functionality by breaking it into manageable parts. In a smart contract like RentSmart, DFD elements include functions (processes), mappings like rooms and agreements (data stores), and users such as landlords and tenants (external entities). DFDs are useful for analyzing and designing systems clearly and efficiently.

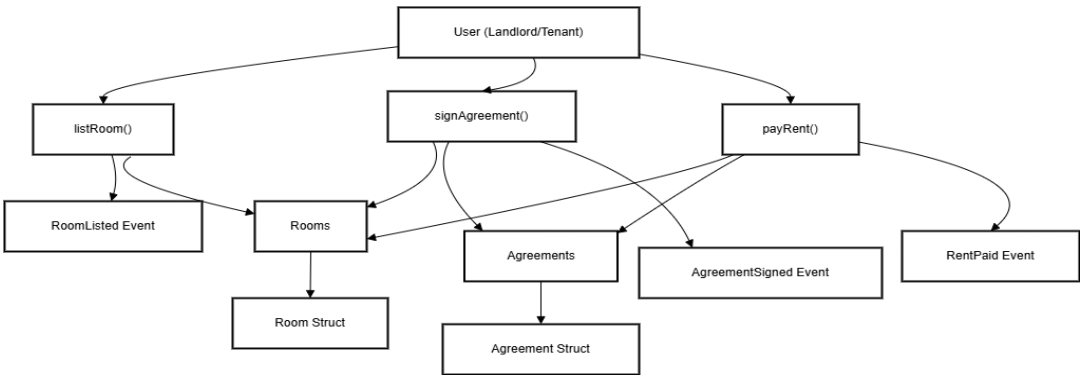


**Figure 4.2 Dataflow Diagram**

This Figure 4.2 shows that a user—either a landlord or a tenant—interacts with the RentSmart smart contract through three primary functions: listRoom(), signAgreement(), and payRent(). The listRoom() function is invoked by a landlord to add a room to the contract, which updates the Rooms mapping that stores room data based on the Room Struct. This action also triggers the RoomListed event. The signAgreement() function is used by a tenant to initiate a lease, which updates both the Rooms and Agreements mappings, relying on the Agreement Struct to define the structure of agreement data. Upon execution, it emits the AgreementSigned event. The payRent() function allows a tenant to pay rent for an active lease. This function also updates both Rooms and Agreements and emits the RentPaid event. Overall, the diagram highlights the flow of data from user actions to smart contract state changes and the corresponding events that provide transparency and traceability.

## 4.4 Usecase Diagram:

A Use Case Diagram is a visual representation of how users interact with a system. It outlines the various functionalities (use cases) that the system provides and shows which user (actor) performs each function. This type of diagram is commonly used in the early stages of system design to understand user requirements and system behavior.

The Figure 4.3 shows the usecase diagram that represents the interaction between the landlord and tenant

**Actors:**

- **Landlord** – A user who lists rooms and receives rent.
- **Tenant** – A user who signs agreements, pays rent, and ends leases.

**Use Cases:**

1. **ListRoom**(Landlord only)- The landlord adds a room to the system.
2. **SignAgreement**(Tenant only)- The tenant signs a rental agreement for a listed room.

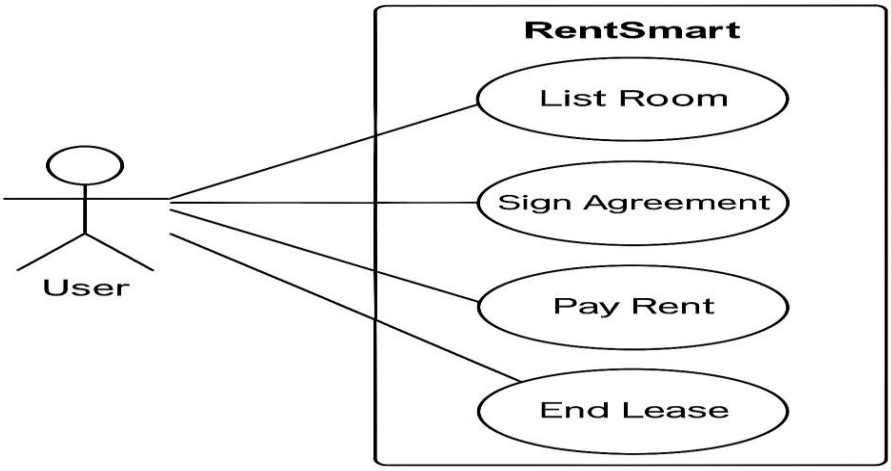


**Figure 4.3 Usecase Diagram**

3. **PayRent** (Tenant only)-The tenant pays rent to the landlord based on the lease terms.

4. **EndLease** (Tenant or System-initiated)- The lease ends either automatically after the duration or by the tenant.

## 4.5 Sequence diagram

The sequence diagram for the RentSmart smart contract, demonstrating the key interactions between the Landlord, Tenant, and the RentSmart Smart Contract. This diagram highlights the step-by-step flow of operations involved in the room rental process on the blockchain.

The Figure 4.4 shown above provides a detailed representation of the interactions between the Landlord, Tenant, and the RentSmart Smart Contract in a decentralized rental platform. This diagram is essential for understanding how various components of the smart contract interact in real-time to facilitate a secure and automated rental process.

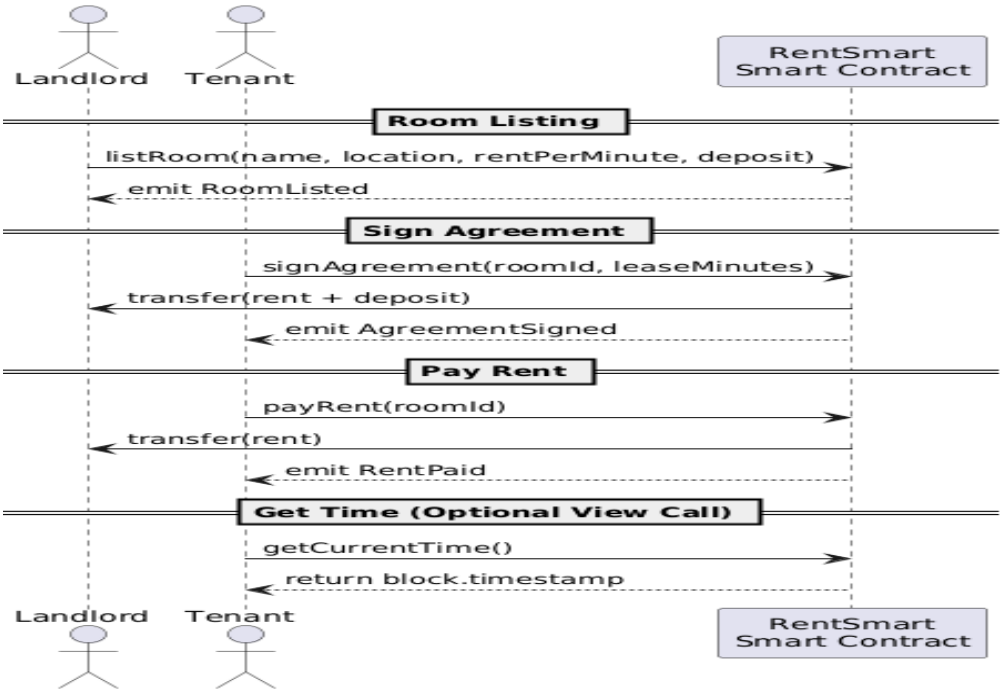


**Figure 4.4 Sequence Diagram**

The process begins with the Landlord calling the listRoom function, where they provide necessary room details such as name, location, rent per minute, and security deposit. Upon successful listing, a RoomListed event is emitted by the smart contract, confirming that the room is available for rent.

Following this, the Tenant interacts with the smart contract by invoking the signAgreement function, specifying the room ID and desired lease duration. Along with the function call, the tenant sends the required amount of Ether (rent + security deposit). Upon validation, the smart contract transfers the amount to the landlord and records a new rental agreement. An AgreementSigned event is then emitted, signaling that the lease agreement has been successfully created and is now active.

During the lease period, the Tenant pays rent by calling the payRent function at specified intervals. The smart contract verifies the payment and transfers the rent to the landlord. A RentPaid event is emitted as confirmation. Additionally, the tenant may optionally call getCurrentTime() to retrieve the current blockchain timestamp, which can help verify lease timings or payment cycle.

# Chapter 5

# System Implementation

## 5.1 Create/List Room

The “Create/List Room” function allows a user to register a new land record on the blockchain.

### Pseudocode:

Function listRoom(name, location, rentPerMinute, securityDeposit):

roomCount ← roomCount + 1

newRoom.id ← roomCount

newRoom.landlord ← caller address

newRoom.name ← name

newRoom.location ← location

newRoom.rentPerMinute ← rentPerMinute

newRoom.securityDeposit ← securityDeposit

newRoom.isVacant ← true

newRoom.currentTenant ← null

newRoom.lastRentPaidAt ← 0

newRoom.agreementId ← 0

rooms[roomCount] ← newRoom

Emit RoomListed(roomCount, caller address)

## 5.2 Sign Agreement

Function signAgreement(roomId, leaseMinutes) payable:

room ← rooms[roomId]

Require room.isVacant == true

Require msg.value >= room.rentPerMinute + room.securityDeposit

Transfer (room.rentPerMinute + room.securityDeposit) to room.landlord

agreementCount ← agreementCount + 1

newAgreement.id ← agreementCount

newAgreement.roomId ← roomId

newAgreement.tenant ← caller address

newAgreement.leaseStart ← current timestamp

newAgreement.leaseDuration ← leaseMinutes * 1 second

newAgreement.active ← true

agreements[agreementCount] ← newAgreement

room.isVacant ← false

room.currentTenant ← caller address

room.lastRentPaidAt ← current timestamp

room.agreementId ← agreementCount

Emit AgreementSigned(agreementCount, caller address)

## 5.3 Payrent

Function payRent(roomId) payable:

room ← rooms[roomId]

agreement ← agreements[room.agreementId]

Require caller == room.currentTenant

Require agreement.active == true

Require msg.value >= room.rentPerMinute

Require current timestamp >= room.lastRentPaidAt + 1 second  // rent due

Transfer room.rentPerMinute to room.landlord

room.lastRentPaidAt ← current timestamp

Emit RentPaid(roomId, caller address, msg.value)

## 5.4 End Lease

Function endLease(agreementId):

agreement ← agreements[agreementId]

room ← rooms[agreement.roomId]

Require caller == room.landlord OR caller == agreement.tenant

Require agreement.active == true

agreement.active ← false

room.isVacant ← true

room.currentTenant ← null

room.agreementId ← 0

Emit LeaseEnded(agreementId)

# Chapter 6

# Testing

Testing is the process of evaluating a system or its components with the intent to find whether it satisfies the specified requirements or not. It includes a set of techniques and methods to identify defects, bugs, performance issues and providing a reliable and quality product. The goal is to identify issues as early as possible and improve the overall quality of the system.

## 6.1 Types of Testing

### 6.1.1 Unit Testing

Unit testing is a type of testing that is used to evaluate the individual units or components of a software system. This type of testing helps ensure that each unit or component of the system is working correctly and is able to perform its intended function.

### 6.1.2 Integration Testing

Integration testing is a type of testing that is used to evaluate how well the different units or components of a software system work together. This type of testing helps to identify and resolve issues related to compatibility, performance, and data flow between the different units or components.

### 6.1.3 Functional Testing

Functional testing is a type of testing that is used to evaluate how well a system or software performs the specific functions or tasks that it is designed to perform. It is done by testing the system or software with various inputs and verifying that the outputs are correct. This type of testing ensures that the system or software is working as intended and is able to perform the functions it was designed to perform.

### 6.1.4 White Box Testing

White box testing, also known as structural testing or glass-box testing, is a type of testing that examines the internal structure and implementation of a software system. It involves testing the code itself and checking that it is functioning correctly and adhering to coding

standards. This type of testing helps to identify and resolve issues related to logic, control flow, and data structures within the system.

### 6.1.5 Black Box Testing

Black box testing, also known as functional testing, is a type of testing that examines the external behavior and interfaces of a software system. It involves testing the system from the user's perspective, without looking at the internal structure or implementation, and checking that it is functioning correctly and meeting the requirements. This type of testing helps to identify and resolve issues related to usability, compatibility, and performance.

### 6.1.6 Testing Outcomes

**Table 1 Summary of Test Cases Passed for RentSmart Smart Contract**

| Test Case ID | Function Tested | Input Parameters | Expected Outcome | Result |
|---|---|---|---|---|
| TC01 | constructor | — | Contract should deploy successfully | Passed |
| TC02 | listRoom | ("Room A", "Location A", 1 ETH/min, 2 ETH deposit) | Room should be listed and event emitted | Passed |
| TC03 | signAgreement | (Room ID: 1, Lease Duration: 5 minutes, Value: 3 ETH) | Agreement should be created and tenant assigned | Passed |
| TC04 | payRent | (Room ID: 1, Value: 1 ETH) | Rent should be transferred and timestamp updated | Passed |
| TC05 | agreementCount | — | Should return total number of agreements (1) | Passed |

The Table 1 shows that all core functionalities of the RentSmart smart contract have been successfully validated through systematic testing. It includes essential operations such as deploying the contract, listing rooms by landlords, signing lease agreements by tenants, making rent payments, and retrieving agreement details. Each function was tested with appropriate inputs and returned the expected results, with all test cases marked as passed. This confirms that the smart contract logic is correctly implemented and operates reliably under typical usage scenarios.

# Results and Discussions

The RentSmart smart contract facilitates decentralized rental agreements on the Ethereum blockchain. It allows landlords to list rooms and tenants to sign lease agreements and pay rent in real time, priced per minute. The contract includes key functionalities like secure payments, lease tracking, and access control via modifiers. Events are emitted for transparency and off-chain tracking. The use of block.timestamp ensures time-based validations, including rent payment intervals. Overall, the contract aims to provide a simple, transparent rental system with automation and minimal manual intervention.

## 7.1 Create/List Room

The transaction demonstrates the successful execution of the listRoom function in the RentSmart smart contract. It adds a new room listing with specified details such as name, location, rent per minute, and security deposit. The transaction was mined in a specific block and consumed a defined amount of gas. This confirms that the smart contract is functioning as intended for room registration.


**Figure 7.1 Snapshot of Create/List Room**

The Figure 7.1 illustrate,

1. **Function Called – listRoom**:The transaction successfully called the listRoom function of the RentSmart contract. The inputs provided were:

- Room name: "Apartment1"
- Location: "Bangalore"
- Rent per minute: 2 (likely in wei)
- Security deposit: 1

2. **Transaction Status – Success**:The status is 0x1, which means the transaction was mined and executed successfully. The transaction was included in block number 22596881.
3. **Gas Usage**:
   - Gas provided: 241208
   - Gas used for execution: 187358
   - Total transaction cost: 209746 gas (including base fees)
4. **Decoded Input Confirmation**:The input data was successfully decoded, confirming that the listRoom function received the correct parameters, as shown in the "decoded input" section. This confirms the function call aligns with expected behavior.

## 7.2 Testing and Validation of Smart Contract

This chapter includes the demonstration and verification of the core functionalities of the RentSmart smart contract through test cases. The image shows transaction logs from a simulated environment, confirming successful deployment and execution of major functions like listRoom, signAgreement, and payRent.

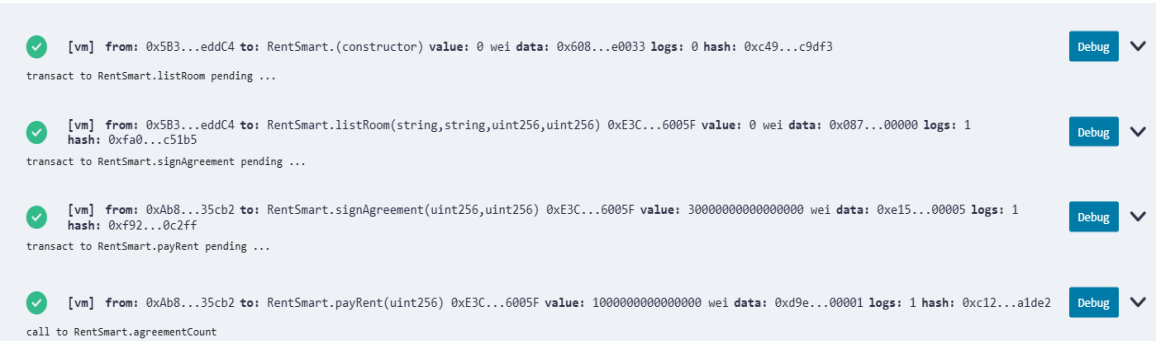


**Figure 7.2 Execution Results of RentSmart Smart Contract Functions**

The Figure 7.2 shows that the core functionalities of the RentSmart smart contract have been successfully tested and executed without any errors. Initially, the contract was deployed with no issues, as indicated by the successful constructor call. Following deployment, the listRoom function was called by the landlord to register a new room, which emitted an event confirming the room listing. Subsequently, a tenant interacted with the contract by invoking the signAgreement function, sending the required rent and security deposit (3 ETH in total), which resulted in a successful lease agreement and event log. The tenant then executed the payRent function by sending 1 ETH to the landlord, and the transaction was processed smoothly with a confirmation event. Finally, the contract's agreementCount variable was accessed to verify the number of active agreements, which indicates proper tracking of lease records. Overall, the figure demonstrates that the RentSmart system is functioning correctly, with each transaction executing as expected and emitting relevant events to track the operations, confirming the smart contract's reliability for rental processes.

# Chapter 8

# Conclusion

The Smart Rental System project successfully demonstrates how blockchain technology, specifically Ethereum smart contracts, can streamline and automate the rental process. The system provides an efficient, transparent, and secure platform for both landlords and tenants to manage room listings, sign agreements, pay rent, and end leases. By eliminating intermediaries, it reduces delays, minimizes disputes, and ensures trust between parties through immutable records and self-executing contract logic. The RentSmart contract was tested and verified through various use case scenarios including room listing, agreement signing, rent payment, and lease termination, all of which functioned as intended.

## Future Enhancements

1. **Lease Termination Logic:**

   Introduce a complete endLease function allowing landlords or tenants to mutually or automatically end a lease and trigger the refund of the security deposit based on predefined conditions.

2. **Dispute Resolution Mechanism:**

   Implement an on-chain/off-chain arbitration system to resolve disputes fairly, ensuring a fallback process if conflicts arise between parties.

3. **User Interface (DApp):**

   Develop a front-end decentralized application (DApp) to provide an intuitive user experience, enabling interaction with the smart contract without requiring coding or command-line interfaces.

4. **Rating and Review System:**

   Allow both tenants and landlords to rate each other post-lease to build reputation and accountability in the ecosystem.

5. **Integration with Oracles:**

Use external data providers (oracles) to fetch real-world data such as location validation, identity verification, or payment conversion rates.

6. **Multi-Signature Approval:**

Add support for multi-party approvals, especially for corporate landlords or joint tenants, to enable collaborative decision-making.

These enhancements aim to broaden the system's capabilities, improve usability, and strengthen trust across a wider range of rental scenarios.

# References

[1]. Q. Xue, W. Guo, and S. Li, “Blockchain-based housing rental system with alliance chain for enhanced security and transparency,” *IEEE Access*, vol. 9, pp. 102345–102356, 2021. doi: 10.1109/ACCESS.2021.3098460.

[2]. Q.-L. Chen, Y. Chen, and Y.-L. Zhu, “Decentralized housing rental system based on blockchain and IPFS,” in *Proc. 5th Int. Conf. on Blockchain Technology and Applications (ICBTA)*, Xi'an, China, 2021, pp. 78–83. doi: 10.1145/3486630.3486654.

[3]. A. S. Proença, D. P. Dias, and P. Ferreira, “A GDPR-compliant blockchain-based rental platform,” in *Proc. 2020 IEEE Int. Conf. on Decentralized Applications and Infrastructures (DAPPS)*, Oxford, UK, 2020, pp. 53–60. doi: 10.1109/DAPPS49028.2020.00014.

[4]. A. Bogner, M. Chanson, and A. Meeuw, “A decentralized sharing app running a smart contract on the Ethereum blockchain,” in *Proc. 6th Int. Conf. on the Internet of Things (IoT)*, Stuttgart, Germany, 2016, pp. 177–178. doi: 10.1145/2991561.2998472.