

# Design and Implementation of a Blockchain-based Smart Rental Contract System

*Xing Shu*

A dissertation submitted in partial fulfilment  
of the requirements for the degree of  
**Master of Science in Cybersecurity**  
of the  
**University of Aberdeen.**



Department of Cyber Security

2024

# Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2024

# Abstract

With the development of blockchain technology, smart contracts have shown tremendous potential in the rental market. Traditional rental methods face numerous security issues, such as information asymmetry, data leakage, and contract tampering. Additionally, rental transactions typically rely on intermediary agencies, which can lead to trust issues. Smart rental contracts manage rental agreements automatically, ensuring transparency and security, and protecting the confidentiality of personal information and the immutability of contracts. This paper describes in detail the design and implementation of a smart rental contract system, focusing on protecting the rights of both parties, reducing trust costs, and mitigating operational risks. Through in-depth research on smart contracts, this study demonstrates their significant potential in enhancing rental market efficiency, reducing costs, increasing transparency, and ensuring information security.

# Acknowledgements

Much stuff borrowed from elsewhere

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Objectives . . . . .	9
1.2	Outline . . . . .	10
<b>2</b>	<b>Literature Review</b>	<b>11</b>
2.1	Blockchain . . . . .	11
2.1.1	Blockchain Technology . . . . .	11
2.1.2	Blockchain technology applications . . . . .	12
2.2	Smart Contracts . . . . .	12
2.3	Ganache . . . . .	13
2.4	MetaMask . . . . .	13
2.5	React.js . . . . .	14
2.6	Web3.js . . . . .	14
2.7	Truffle . . . . .	15
<b>3</b>	<b>System Functional Module Description</b>	<b>16</b>
3.1	Connecting the MetMask . . . . .	16
3.1.1	Applicable Personnel . . . . .	16
3.1.2	Preconditions . . . . .	16
3.1.3	Description . . . . .	16
3.1.4	Postconditions . . . . .	16
3.1.5	Main Flow . . . . .	16
3.1.6	Alternative Flows . . . . .	16
3.1.7	Frontend description . . . . .	17
3.1.8	Interfaces and Interactions: . . . . .	17
3.1.9	Security Analysis . . . . .	17
3.2	Contract Activation . . . . .	17
3.2.1	Applicable Personnel . . . . .	17
3.2.2	Preconditions . . . . .	17
3.2.3	Description . . . . .	17
3.2.4	Postconditions . . . . .	18
3.2.5	Main Flow . . . . .	18
3.2.6	Alternative Flows . . . . .	18
3.2.7	Frontend description . . . . .	18

3.2.8	Interfaces and Interactions: . . . . .	18
3.2.9	Security Analysis . . . . .	18
3.3	Landlord Dashboard . . . . .	19
3.3.1	Applicable Personnel . . . . .	19
3.3.2	Preconditions . . . . .	19
3.3.3	Description . . . . .	19
3.3.4	Postconditions . . . . .	19
3.3.5	Main Flow . . . . .	19
3.3.6	Frontend description . . . . .	20
3.3.7	Interfaces and Interactions: . . . . .	20
3.3.8	Security Analysis . . . . .	20
3.4	Add and Modify Room Status . . . . .	20
3.4.1	Applicable Personnel . . . . .	20
3.4.2	Preconditions . . . . .	20
3.4.3	Description . . . . .	21
3.4.4	Postconditions . . . . .	21
3.4.5	Main Flow . . . . .	21
3.4.6	Frontend description . . . . .	22
3.4.7	Interfaces and Interactions: . . . . .	22
3.4.8	Security Analysis . . . . .	22
3.5	View and Modify Tenant Request Lists . . . . .	22
3.5.1	Applicable Personnel . . . . .	22
3.5.2	Preconditions . . . . .	22
3.5.3	Description . . . . .	22
3.5.4	Postconditions . . . . .	23
3.5.5	Main Flow . . . . .	23
3.5.6	Frontend description . . . . .	23
3.5.7	Interfaces and Interactions: . . . . .	23
3.5.8	Security Analysis . . . . .	23
3.6	View Rental Information by Address . . . . .	23
3.6.1	Applicable Personnel . . . . .	23
3.6.2	Preconditions . . . . .	23
3.6.3	Description . . . . .	24
3.6.4	Postconditions . . . . .	24
3.6.5	Main Flow . . . . .	24
3.6.6	Frontend description . . . . .	24
3.6.7	Interfaces and Interactions: . . . . .	24
3.6.8	Security Analysis . . . . .	24
3.7	Tenant Registration . . . . .	25
3.7.1	Applicable Personnel . . . . .	25
3.7.2	Preconditions . . . . .	25
3.7.3	Description . . . . .	25

3.7.4	Postconditions . . . . .	26
3.7.5	Main Flow . . . . .	26
3.7.6	Frontend description . . . . .	26
3.7.7	Interfaces and Interactions: . . . . .	26
3.7.8	Security Analysis . . . . .	26
3.8	Tenant Dashboard . . . . .	26
3.8.1	Applicable Personnel . . . . .	26
3.8.2	Preconditions . . . . .	26
3.8.3	Description . . . . .	27
3.8.4	Postconditions . . . . .	27
3.8.5	Main Flow . . . . .	27
3.8.6	Frontend description . . . . .	28
3.8.7	Interfaces and Interactions: . . . . .	28
3.8.8	Security Analysis . . . . .	28
3.9	Booking Room . . . . .	28
3.9.1	Applicable Personnel . . . . .	28
3.9.2	Preconditions . . . . .	28
3.9.3	Description . . . . .	29
3.9.4	Postconditions . . . . .	30
3.9.5	Main Flow . . . . .	31
3.9.6	Frontend description . . . . .	31
3.9.7	Interfaces and Interactions: . . . . .	31
3.9.8	Security Analysis . . . . .	31
3.10	Tenant Payment . . . . .	31
3.10.1	Applicable Personnel . . . . .	31
3.10.2	Preconditions . . . . .	31
3.10.3	Description . . . . .	32
3.10.4	Postconditions . . . . .	32
3.10.5	Main Flow . . . . .	32
3.10.6	Frontend description . . . . .	32
3.10.7	Interfaces and Interactions: . . . . .	33
3.10.8	Security Analysis . . . . .	33
3.11	Submit Request . . . . .	33
3.11.1	Applicable Personnel . . . . .	33
3.11.2	Preconditions . . . . .	33
3.11.3	Description . . . . .	33
3.11.4	Postconditions . . . . .	34
3.11.5	Main Flow . . . . .	34
3.11.6	Frontend description . . . . .	34
3.11.7	Interfaces and Interactions: . . . . .	34
3.11.8	Security Analysis . . . . .	34

<b>4</b>	<b>System Deployment</b>	<b>35</b>
4.1	System Deployment . . . . .	35
4.1.1	Deployment Environment Configuration . . . . .	35
4.1.2	Deployment Steps . . . . .	35
<b>5</b>	<b>System Security Analysis</b>	<b>39</b>
5.1	Information Reliability Analysis . . . . .	39
5.1.1	Data Validation and Verification . . . . .	39
5.1.2	Redundancy and Consistency . . . . .	39
5.1.3	Trust Mechanisms . . . . .	39
5.1.4	Results and Discussion . . . . .	39
5.2	Information Security Analysis . . . . .	40
5.2.1	Encryption and Data Protection . . . . .	40
5.2.2	Access Control . . . . .	40
5.2.3	Monitoring and Incident Response . . . . .	40
5.2.4	Results and Discussion . . . . .	40
5.3	Information Immutable Analysis . . . . .	40
5.3.1	Blockchain Immutability . . . . .	40
5.3.2	Cryptographic Hashing . . . . .	41
5.3.3	Smart Contract Immutability . . . . .	41
<b>6</b>	<b>Challenges and Future Outlook</b>	<b>42</b>
6.1	Challenges . . . . .	42
6.1.1	Technical Challenges . . . . .	42
6.1.2	Operational Challenges . . . . .	42
6.1.3	Market Challenges . . . . .	42
6.2	Future Outlook . . . . .	43
6.2.1	Technological Advancements . . . . .	43
6.2.2	Operational Improvements . . . . .	43
6.2.3	Market Strategies . . . . .	43
<b>7</b>	<b>Conclusions</b>	<b>44</b>



## Chapter 1

# Introduction

As the global rental market continues to grow, it has become a significant and vital economic sector worldwide. However, the traditional rental market faces numerous challenges, including information asymmetry, difficulties in contract enforcement, high transaction costs, and frequent disputes. Rental transactions typically depend on intermediary agencies, which not only increase costs but also can lead to trust issues. Additionally, the efficiency and transparency of contract enforcement need improvement, particularly in areas such as rent payments and contract amendments, where traditional methods are often cumbersome and inefficient. More importantly, the traditional rental market has significant risks in data security and privacy protection, making it vulnerable to fraud, data breaches, and unauthorized access.

The advent of blockchain technology offers new solutions to these problems. Blockchain's decentralized, immutable, and transparent nature provides inherent advantages for smart contracts in terms of automated execution and trust mechanisms. Smart rental contracts can automate the execution of rental agreements, handle rent payments automatically, and effectively protect the rights of both parties, thereby enhancing the efficiency and transparency of the rental market and reducing the operational risks associated with intermediaries. Furthermore, blockchain's encryption technology and distributed storage ensure the security and privacy of rental transaction data, effectively preventing data tampering and unauthorized access, and reducing the risk of information leakage. Through these security mechanisms, smart rental contracts not only increase transaction transparency and efficiency but also offer higher security guarantees for both parties involved.

## 1.1 Objectives

This study aims to comprehensively explore the application and potential advantages of smart rental contracts in the modern rental market. With the continuous advancement of blockchain technology, smart contracts, as a new automated management tool, can significantly enhance the efficiency and security of the rental market. This paper also explores how to use this tool more efficiently and maximize its security benefits.

We designed and implemented a smart rental contract system. Through in-depth research on smart rental contracts, we aim to demonstrate their specific application scenarios and technical implementation processes in the rental market. We will explore how smart contracts can improve overall market efficiency by automating rental agreement execution, ensuring transaction transparency, protecting user privacy, and reducing operational risks.

In summary, this research is not only an exploration and implementation of smart rental

contract technology but also a comprehensive demonstration of its advantages in real-world applications, such as efficiency improvement, cost reduction, and increased transparency. Through this research, we hope to provide rental market participants with a more secure, efficient, and transparent transaction method, thereby supporting a safer digital transformation of the rental market.

## 1.2 Outline

This dissertation is organised into 6 chapters, as described below

Chapter 1: Introduction - This chapter introduces the topic area and presents the problem statement. It sets the context for the dissertation, outlining the significance of the study and the objectives it aims to achieve.

Chapter 2: Literature Review - This chapter reviews the terminology and concepts related to the technologies used in this project. It provides insights gained from various papers and includes summaries of different projects that have adopted these technologies to address practical problems. The review aims to establish a theoretical foundation and highlight existing gaps that this dissertation intends to fill.

Chapter 3: System Functional Description - This chapter provides a comprehensive description of each functional module of the system. It details the architecture, design, and functionality of the modules, explaining how they interact to achieve the system's objectives.

Chapter 4: System Deployment - This chapter describes some softwares that you need to download for the deployment, and the configuration of the system deployment environment.

Chapter 5: Security Analysis - This chapter presents a thorough analysis of the system's security. It includes evaluations of information reliability, information security, and information immutability, along with penetration testing results. The analysis aims to identify potential vulnerabilities and propose measures to mitigate them.

Chapter 6: Challenges and Future Outlook - This chapter discusses the challenges encountered during the development and implementation of the system. It explores future developments and directions, presenting the achieved results with both quantitative and qualitative analysis. This chapter also includes recommendations for addressing identified challenges and improving the system.

Chapter 7: Conclusions - The final chapter provides a summary and critical analysis of the work realized in this dissertation. It highlights the key findings, evaluates the significance of the results, and suggests potential future research directions. The conclusions aim to reflect on the dissertation's contributions and its impact on the field.

## Chapter 2

# Literature Review

### 2.1 Blockchain

Blockchain refers to a decentralized, distributed ledger technology that records transactions across multiple computers so that the record cannot be altered retroactively without altering all subsequent blocks and the consensus of the network [1]. Initially conceptualized for Bitcoin by an anonymous entity known as Satoshi Nakamoto in 2008, blockchain has since been applied to various domains beyond cryptocurrencies, including finance, supply chain management, and health-care [2].

A blockchain consists of a series of blocks, each containing a cryptographic hash of the previous block, a timestamp, and transaction data. This structure ensures data integrity and transparency, as every transaction is verifiable and traceable through the entire chain [3]. The decentralized nature of blockchain eliminates the need for a central authority, thereby reducing the risk of corruption and fraud [4].

#### 2.1.1 Blockchain Technology

The primary components of blockchain technology include:

- **Decentralization:** Unlike traditional centralized systems where a single entity has control, blockchain operates on a peer-to-peer network. Each participant (node) maintains a copy of the entire blockchain and verifies new transactions through a consensus mechanism [5].
- **Transparency:** All transactions on the blockchain are visible to all participants. This transparency fosters trust among users, as they can independently verify and audit the transactions [6].
- **Immutability:** Once a block is added to the blockchain, it cannot be altered or deleted. This immutability is achieved through cryptographic hashing and consensus algorithms, which ensure that any attempt to change a block would require altering all subsequent blocks, a computationally impractical task [7].
- **Consensus Mechanisms:** These are protocols used by blockchain networks to agree on the validity of transactions. Common consensus mechanisms include Proof of Work (PoW), Proof of Stake (PoS), and Practical Byzantine Fault Tolerance (PBFT). These mechanisms prevent double-spending and ensure the security and reliability of the blockchain [8].

- **Smart Contracts:** These are self-executing contracts with the terms directly written into code. They automatically enforce and execute contract terms when predefined conditions are met, reducing the need for intermediaries and enhancing efficiency and trust [9].

### 2.1.2 Blockchain technology applications

Blockchain technology has several applications, including:

- **Cryptocurrencies:** Blockchain's first and most well-known application is in cryptocurrencies like Bitcoin and Ethereum, where it provides a secure and transparent way to conduct transactions without a central authority [10].
- **Supply Chain Management:** Blockchain can improve supply chain transparency and efficiency by providing a tamper-proof record of the entire product lifecycle, from production to delivery [11].
- **Healthcare:** In healthcare, blockchain can be used to securely store and share patient records, ensuring data integrity and privacy while enabling seamless access to medical history across different healthcare providers [12].

## 2.2 Smart Contracts

Smart contracts are self-executing contracts with the terms of the agreement directly written into lines of code. These contracts exist across a distributed, decentralized blockchain network, ensuring transparency, traceability, and security [13]. The concept of smart contracts was first proposed by Nick Szabo in 1994, long before blockchain technology was established, to enable trusted transactions without the need for third parties [14].

A smart contract operates by following simple "if/when...then..." statements that are written into code and executed automatically when predetermined conditions are met. This automation reduces the need for intermediaries, cuts down on costs, and decreases the potential for errors and fraud [15].

The primary components of smart contracts include:

- **Code and Data:** The contract's code defines the rules and penalties around the agreement, while the data stores the state of the contract and any necessary information regarding its execution [16].
- **Decentralization:** These contracts are stored and executed on a decentralized blockchain, which ensures that no single party has control over the contract and that all actions are transparent and verifiable by all parties involved [17].
- **Immutability:** Once deployed on the blockchain, smart contracts cannot be altered. This immutability ensures that the contract terms are enforced exactly as written, providing a high level of trust and reliability [18].
- **Autonomy:** Smart contracts operate autonomously, meaning they automatically execute the agreed-upon terms without human intervention once the conditions are met [19].
- **Trust and Transparency:** The decentralized nature and openness of the blockchain ensure that all parties can trust the contract's execution and verify its terms and outcomes [20].

Smart contracts have several applications, including:

- **Financial Services:** In finance, smart contracts can automate transactions like loan agreements, insurance claims, and securities trading, reducing the need for intermediaries and increasing efficiency [21].
- **Supply Chain Management:** Smart contracts can automate and verify the transfer of goods and payments, ensuring transparency and reducing the risk of fraud in supply chain processes [22].
- **Real Estate:** Smart contracts can streamline real estate transactions by automating processes like property transfers, lease agreements, and escrow services, making transactions faster and more secure [23].
- **Healthcare:** In healthcare, smart contracts can manage patient consent, automate billing processes, and ensure the secure and private sharing of medical data [24].

## 2.3 Ganache

Ganache is a personal blockchain for Ethereum development that you can use to deploy contracts, develop your applications, and run tests [25]. It is part of the Truffle Suite of Ethereum development tools and provides an easy-to-use interface for managing and interacting with an Ethereum blockchain locally [26]. Ganache simulates the features of the Ethereum blockchain, allowing developers to test their smart contracts and decentralized applications (dApps) in a safe and controlled environment before deploying to the main network [27].

The primary features of Ganache include:

- **Personal Blockchain:** Ganache runs a personal Ethereum blockchain on your local machine, which is used to deploy contracts, run tests, and execute commands [28].
- **Graphical User Interface (GUI):** Ganache offers a GUI that provides a visual representation of the blockchain, including blocks, transactions, and accounts, making it easier to manage and debug smart contracts [29].
- **Fast and Easy Setup:** Setting up Ganache is straightforward, requiring minimal configuration. It provides instant mining of transactions, eliminating the need to wait for block confirmations [30].
- **Customizable Parameters:** Developers can customize various blockchain parameters, such as block time, gas limits, and network ID, to mimic different network conditions [31].
- **Logs and Debugging:** Ganache provides detailed logs and debugging tools, allowing developers to inspect the state of the blockchain and troubleshoot issues in their smart contracts [32].

## 2.4 MetaMask

MetaMask is a cryptocurrency wallet and gateway to blockchain applications. It allows users to manage their Ethereum-based assets and interact with decentralized applications (dApps) directly from their web browsers [33]. Available as a browser extension and mobile app, MetaMask simplifies the process of using dApps by providing a user-friendly interface and secure key management [34].

The primary features of MetaMask include:

- **Wallet:** MetaMask functions as a digital wallet, enabling users to store, send, and receive Ethereum and ERC-20 tokens [35].

- **dApp Browser:** MetaMask acts as a bridge between the user's browser and the Ethereum blockchain, allowing seamless interaction with dApps without running a full Ethereum node [36].
- **Secure Key Management:** MetaMask securely stores private keys and seed phrases locally on the user's device, ensuring that users have full control over their funds [37].
- **Network Switching:** Users can easily switch between different Ethereum networks (e.g., Mainnet, Ropsten, Kovan) and custom networks, making it versatile for development and testing [38].
- **Custom Tokens:** MetaMask allows users to add custom tokens to their wallet, providing flexibility in managing various types of digital assets [39].

## 2.5 React.js

React.js is a popular JavaScript library developed by Facebook for building user interfaces, especially for single-page applications. It allows developers to create reusable UI components, which makes the development process more efficient and organized. React.js is known for its performance due to the use of a virtual DOM, which optimizes rendering and updates to the real DOM.

The primary features of React.js include:

- **Component-Based Architecture:** React.js uses a component-based architecture, where the UI is divided into independent, reusable components. Each component manages its own state and can be composed to build complex user interfaces [40].
- **Virtual DOM:** React.js employs a virtual DOM to optimize updates and rendering. Changes to the UI are first applied to the virtual DOM, which then determines the most efficient way to update the real DOM, resulting in improved performance [41].
- **JSX Syntax:** React.js uses JSX, a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript. This makes it easier to create and visualize the structure of the UI components [42].
- **One-Way Data Binding:** React.js uses one-way data binding, where data flows in a single direction from parent to child components. This makes the data flow more predictable and easier to debug [43].
- **React Hooks:** React introduced Hooks in version 16.8, allowing developers to use state and other React features without writing a class. Hooks simplify the management of component logic and state [44].

## 2.6 Web3.js

Web3.js is a collection of libraries that allows you to interact with a local or remote Ethereum blockchain. It provides a comprehensive JavaScript API for developers to create and manage decentralized applications (dApps) that run on the Ethereum network [45]. Web3.js facilitates communication with the Ethereum blockchain, enabling tasks such as retrieving user accounts, sending transactions, and interacting with smart contracts [46].

The primary features of Web3.js include:

- **Ethereum Interaction:** Web3.js provides a set of APIs for interacting with the Ethereum blockchain, including methods for querying blockchain data, sending transactions, and managing accounts [47].
- **Smart Contract Integration:** Developers can use Web3.js to deploy and interact with smart contracts, making it a vital tool for building dApps [48].
- **Event Handling:** Web3.js supports event subscriptions, allowing developers to listen for events emitted by smart contracts and react accordingly [49].
- **Compatibility:** Web3.js is compatible with various Ethereum clients and networks, including Geth, Parity, and Infura, ensuring broad applicability in different development environments [50].
- **Ease of Use:** With its extensive documentation and active community, Web3.js is relatively easy to learn and integrate into Ethereum-based projects [51].

## 2.7 Truffle

Truffle is a development framework for Ethereum that provides a suite of tools for developers to build, test, and deploy smart contracts. It simplifies the process of managing the entire lifecycle of decentralized applications (DApps) on the Ethereum blockchain.

The primary features of Truffle include:

- **Smart Contract Compilation:** Truffle offers powerful compilation features for Solidity smart contracts. It automates the process of compiling and linking contracts, making it easier for developers to manage their codebase [52].
- **Testing Framework:** Truffle includes a built-in testing framework that supports both JavaScript and Solidity tests. This allows developers to write comprehensive tests for their smart contracts, ensuring reliability and security [53].
- **Deployment Automation:** Truffle automates the deployment of smart contracts to various Ethereum networks. This includes both local test networks and public testnets/mainnets, streamlining the deployment process [54].
- **Network Management:** Truffle manages different Ethereum networks, allowing developers to easily switch between networks and deploy contracts in various environments. This feature is crucial for testing and development [55].
- **Interactive Console:** Truffle provides an interactive console for direct interaction with the Ethereum network. Developers can use this console to execute commands and scripts, making debugging and testing more efficient [56].

## Chapter 3

# System Functional Module Description

### 3.1 Connecting the MetMask

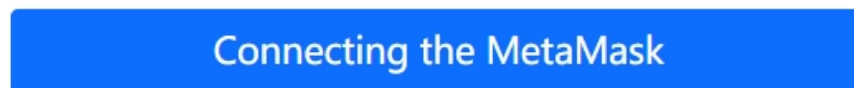
#### 3.1.1 Applicable Personnel

- landlord
- tenant

#### 3.1.2 Preconditions

- The host has deployed the system project
- browser has downloaded and installed the MetaMask plugin
- your MetaMask is connected with one Ethereum account

#### 3.1.3 Description



**Figure 3.1:** the button of connecting MetaMask

Click the button to open the MetaMask software to connect to one of your Ethereum accounts

#### 3.1.4 Postconditions

- - If the connection is successful, the link will be redirected to another page.
- - If the connection is fails, an error message is displayed, and you are prompted to re-connect your MetaMask again.

#### 3.1.5 Main Flow

- **step 1:** Open the website link.
- **step 2:** Click the connecting MetaMask button.
- **step 3:** Agree to the request on MetaMask

#### 3.1.6 Alternative Flows

null



### 3.1.7 Frontend description

When the link is opened, the front-end will first use the web3 package to determine whether the user has linked to MetaMask, and if it does not, it will jump to the functional webpage.

### 3.1.8 Interfaces and Interactions:

After connecting to MetaMask through web3 library, obtain the connected account information by using the method `web3.eth.requestAccounts()`

### 3.1.9 Security Analysis

- **Authentication:** Ensures that only Ethereum accounts can access this site.
- **Data Privacy:** User privacy is protected throughout the operation.

## 3.2 Contract Activation

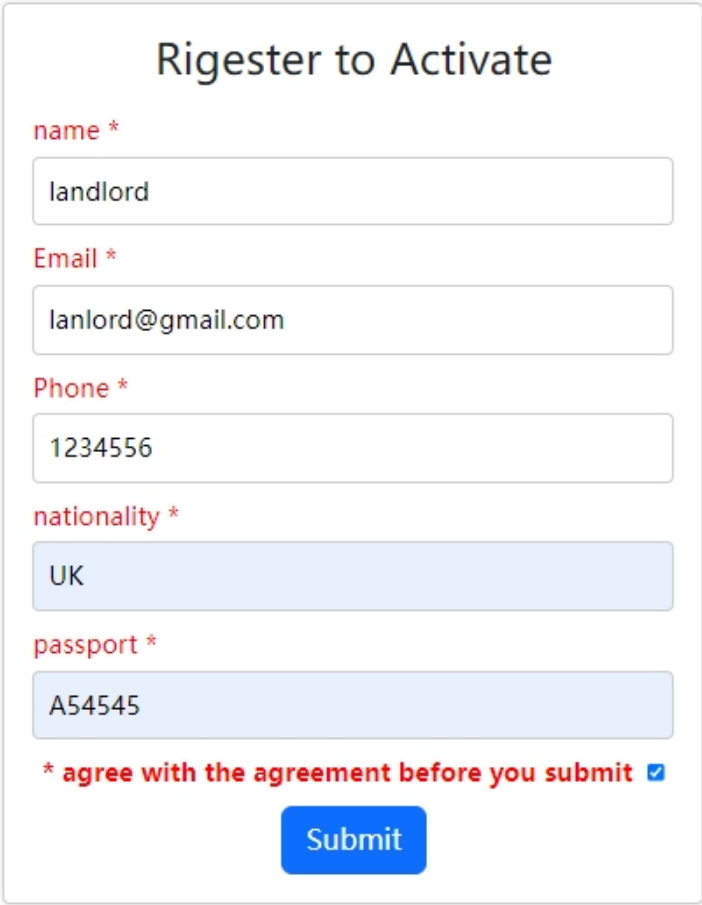
### 3.2.1 Applicable Personnel

- landlord

### 3.2.2 Preconditions

- The landlord has already connected his account via MetaMask.
- The landlord has a sufficient balance in his Ethereum account

### 3.2.3 Description



**Rigester to Activate**

name \*

landlord

Email \*

landlord@gmail.com

Phone \*

1234556

nationality \*

UK

passport \*

A54545

\* agree with the agreement before you submit ☒

Submit

**Figure 3.2:** the image of activating webpage

landlord fills in the relevant information including name, email, phone, nationality, passport, landlord read and agree with the landlord agreement to activate all functions of this contract.

#### 3.2.4 Postconditions

- - If the landlord successfully registers the information and activates the contract, the browser link redirects to the landlord home page.
- - If the landlord fails, there may be a problem with the information filled in, and you need to re-activate the smart contract according to the prompt.

#### 3.2.5 Main Flow

- **step 1:** Fill in each information as required on the activation page.
- **step 2:** click the red letter at the bottom to view the agreement contents.
- **step 3:** click the checkbox button to agree to the agreement.
- **step 4:** click the Submit button.
- **step 5:** Agree to the request on MetaMask

#### 3.2.6 Alternative Flows

null

#### 3.2.7 Frontend description

After clicking the submit button, the front end will judge whether the user has filled in all the information, and at the same time will judge that the information conforms to the corresponding format, if one of them does not meet, it can not submit and will display the corresponding promotion information. In addition, the front-end will also determine whether the user has clicked the checkbox through the agreement terms, if there is no consent, there is no way to submit successfully.

#### 3.2.8 Interfaces and Interactions:

the system uses the contract method 'isActivate()' to check if the contract is activated. If not activated, it redirects to the activation page. The landlord needs to fill in fields such as name, email, phone, nationality, and passport. After completing and submitting, the 'addTenant()' method is called to register their information into the contract. Once the landlord's information is registered, the contract code logic will default to the contract being activated.

- - 'isActivate()': Self-evaluated gas, accessible only by the landlord.
- - 'addTenant()': Self-evaluated gas, accessible by any account.

#### 3.2.9 Security Analysis

Once the landlord adds their account address and agreement content and deploys it, the contract code cannot be modified. Anyone can view the contract content, addressing the transparency issue of traditional contracts. Although the landlord needs to provide some personal information, this information will not be displayed in the contract. Only the landlord can view their information, and once confirmed, it cannot be modified. This not only resolves potential disputes but also addresses data leakage and unauthorized access threats.

### 3.3 Landlord Dashboard

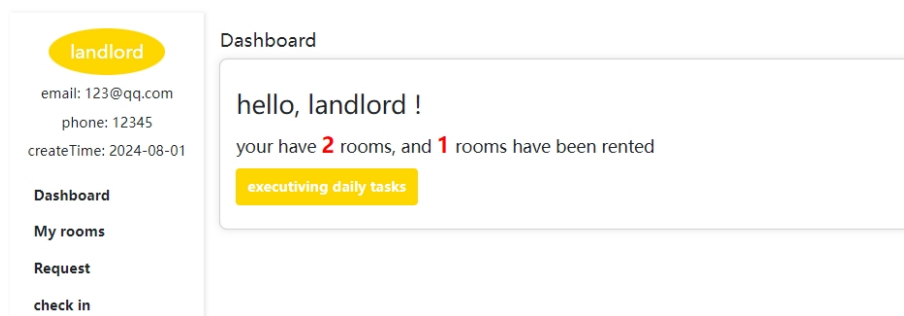
#### 3.3.1 Applicable Personnel

- landlord

#### 3.3.2 Preconditions

- The landlord has already connected his account via MetaMask.
- The landlord has registered the personal information and activated this contract
- landlord has initialized some rooms

#### 3.3.3 Description



**Figure 3.3:** the image of landlord dashboard

the landlord can view personal information, The landlord can view the total number of rooms, the number of available rooms, and the number of booked rooms on the dashoard. By clicking the button, they can perform daily functions to update the contract status (since the contract cannot automatically update room status, the landlord needs to perform this function themselves. The agreement signed have already specify the consequences if the landlord fails to perform this function).

#### 3.3.4 Postconditions

- - If the status of the room is updated, the landlord can immediately know about this situation.
- - If the landlord performs a routine task function, the state of the room may change.
- - If the landlord does not perform the daily tasks, it may bring corresponding losses to the landlord

#### 3.3.5 Main Flow

- **step 1:** open the system website link, the front-end switches to the dashboard page based on the corresponding logic.
- **step 2:** View the corresponding room status and personal information on the dashboard.
- **step 3:** click the executiving daily tasks button to execute the daily task.

### 3.3.6 Frontend description

The front end determines the availability of each room based on the 'isAvailable' attribute and renders this information on the page. And the front end get the current time by the Date.now() of Javascript Object Method.

### 3.3.7 Interfaces and Interactions:

- **hasRegister():** No gas required. Accessible by any users. Need to pass parameters of address(your Ethereum account address). Determine whether the information has been registered by the address, if registered, the corresponding personal information is returned.
- **getAllRooms():** No gas required. Accessible by any users. Not need to pass any parameters. Get all room information.
- **dailyAction():** No gas required. Accessible by the landlord. Need to pass parameters of curTime ( the current time). Updates the corresponding rental and room status based on the comparison between the curTime and the 'endTime' attribute under the rental information.

### 3.3.8 Security Analysis

Due to the immutability of smart contracts, the rental information still needs to be updated daily by the landlord by passing the current time, which does not affect transparency and security. The landlord updates only the current time to update the status, which does not affect the interests of both parties (however, if the landlord fails to update the daily tasks, it may affect the landlord's interests, which is specified during contract activation). Thus, the entire process is transparent and secure. Moreover, this daily task can only be accessed by the landlord and can only be called once a day, preventing misuse and spam transactions, and avoiding the consumption of network resources.

## 3.4 Add and Modify Room Status

### 3.4.1 Applicable Personnel

- landlord

### 3.4.2 Preconditions

- The landlord has already connected his account via MetaMask.
- The landlord has registered the personal information and activated this contract

### 3.4.3 Description

The figure displays three panels illustrating the room status modification process. The first panel shows a room card for '400 x 200' with a red 'Not available!' stamp. The second panel shows the same room card with 'close' and 'open' buttons. The third panel shows a form to add a new room with fields for Address, Block, Room type, Price, and description, and an 'addRoom' button.

**Figure 3.4:** the image of modifying rooms state

The landlord can modify the room status to available or unavailable based on the real situation and can add a new room as needed. However, to ensure security, rooms cannot be deleted; the landlord can only mark rooms as unavailable. Therefore, the landlord needs to add each room cautiously.

### 3.4.4 Postconditions

- - If clicking the close button, the corresponding room will be not available.
- - If clicking the open button, the corresponding room will be available.
- - If clicking the addRoom button and get the alert successfully, will be redirected to the homepage.
- - If clicking the addRoom button and an error message is displayed, you need to try again based on the prompt.

### 3.4.5 Main Flow

- **step 1:** If you want to get the room available, click the open button inside of corresponding room.
- **step 2:** If you want to get the room not available, click the close button inside of corresponding room.

- **step 3:** If you want to add new room, filling the correct information of address, block, room type, description, price, click the addRoom button.

### 3.4.6 Frontend description

The front end will determine whether the room information filled in by the landlord is correct and proceed to the next step.

### 3.4.7 Interfaces and Interactions:

- **getAllRooms():** No gas required. Accessible by any users. Not need to pass any parameters. Get all room information.
- **updateRoomState():** Automatically evaluated gas. Accessible by the landlord. Need to pass parameters of roomId and state. If state is true, the room's state will be available, if false, the room's state will be not available.
- **addRoom():** Automatically evaluated gas. Accessible by the landlord. Need to pass parameters of 'rent', 'addressInfo', 'location', and 'description'. If all parameters are correct, will have a new room.

### 3.4.8 Security Analysis

To ensure the immutability of rooms, once a room is created, it will be permanently stored in the contract. The landlord can only modify the room status. The parameters passed to the contract will be validated to ensure they are valid before executing the corresponding logic, ensuring the contract methods cannot be arbitrarily modified. Moreover, only the landlord can access the contract methods, effectively preventing unauthorized access risks while ensuring transparency.

## 3.5 View and Modify Tenant Request Lists

### 3.5.1 Applicable Personnel

- landlord

### 3.5.2 Preconditions

- The landlord has already connected his account via MetaMask.
- The landlord has registered the personal information and activated this contract
- One customer at least has rented a room and submitted a request at least.

### 3.5.3 Description

Request List

RoomInfo	requestType	description	createTime	state	action
Aberdeen, Causeway View 501D	category	give you descriptions	2024-08-01	solved	solved time: 2024-08-01
Aberdeen, Causeway View 501D	Category2	description2	2024-08-01	solving	handling the request

Figure 3.5: request<sub>list</sub>

The landlord can view the history list of tenant requests, each including 'roomInfo', 'requestType', 'description', 'createTime', and 'state' attributes. The landlord can resolve the request offline based on the information. After resolving the request, the landlord can mark it as resolved. For resolved requests, the landlord can view the resolution time.

### 3.5.4 Postconditions

- - If clicking the handing the request button, the state of the corresponding request will be changed to solved, and the solved time will be displayed in the action column.

### 3.5.5 Main Flow

- **step 1:** Click the Request List tab to check the request lists.
- **step 2:** Deal with the request in person based on the information of the request.
- **step 3:** Click the handling the request button after you have already handled the request in person.

### 3.5.6 Frontend description

The front end display the request using the bootstrap table component, and get the current time by the Date.now() of Javascript Object Method.

### 3.5.7 Interfaces and Interactions:

- **getAllRequest():** No gas required. Accessible by landlord. Not need to pass any parameters. Get all request information.
- **modifyRequest():** Automatically evaluated gas. Accessible by the landlord. Need to pass parameters of 'requestId' and 'endTime'. This method will set the 'isResolved' attribute to true and update the 'endTime', which is the request resolution time.

### 3.5.8 Security Analysis

When a customer encounters an issue related to the room, they can submit a problem description. These descriptions do not include personal privacy information and are stored in the contract, ensuring data immutability. This not only ensures privacy but also resolves potential future disputes.

## 3.6 View Rental Information by Address

### 3.6.1 Applicable Personnel

- lanlord

### 3.6.2 Preconditions

- The landlord has already connected his account via MetaMask.
- The landlord has registered the personal information and activated this contract
- One customer has rented a room but not check in so far.

### 3.6.3 Description

Dashboard

Searching the tenancy information using address

0xA792F6F4776c93e3e65443f04CeA8f184F908853

Search

**The Result**

RoomInfo: **Aberdeen, Causeway View 501D**

Price: **£125.00 per week**

Check In Date: **2024-08-01**

Check Out Date: **2024-08-29**

total prices: **£500**

**have paid all rent!**

**Figure 3.6:** Get rent information by Address

The landlord can view the rental information of an address provided by the tenant and process the corresponding offline procedures based on this information.

### 3.6.4 Postconditions

- - If enter an correct address, will display the corresponding tenancy information.
- - If enter an wrong address, will display null.

### 3.6.5 Main Flow

- **step 1:** Click the Check in tab to get the web page.
- **step 2:** Enter the address that the tenant provide.
- **step 3:** Click the Search button to get the corresponding information.

### 3.6.6 Frontend description

The front-end will determine whether the format of the entered address is correct. If the format is correct, the front-end will display the corresponding information according to the information returned by the interface.

### 3.6.7 Interfaces and Interactions:

- **getTenantRentInfoByAd():** Automatically evaluated gas. Accessible by landlord. Need to pass any parameters of tenantAddress. Get the rental information by the tenantAddress. If the address is not available, it will return empty.

### 3.6.8 Security Analysis

After a tenant rents a house through a smart contract, they will check in in person using their account address. The landlord only verifies whether the user has signed the rental information



through the smart contract using this address, without asking for other details. This process not only protects the personal privacy of both parties but also increases efficiency.

## 3.7 Tenant Registration

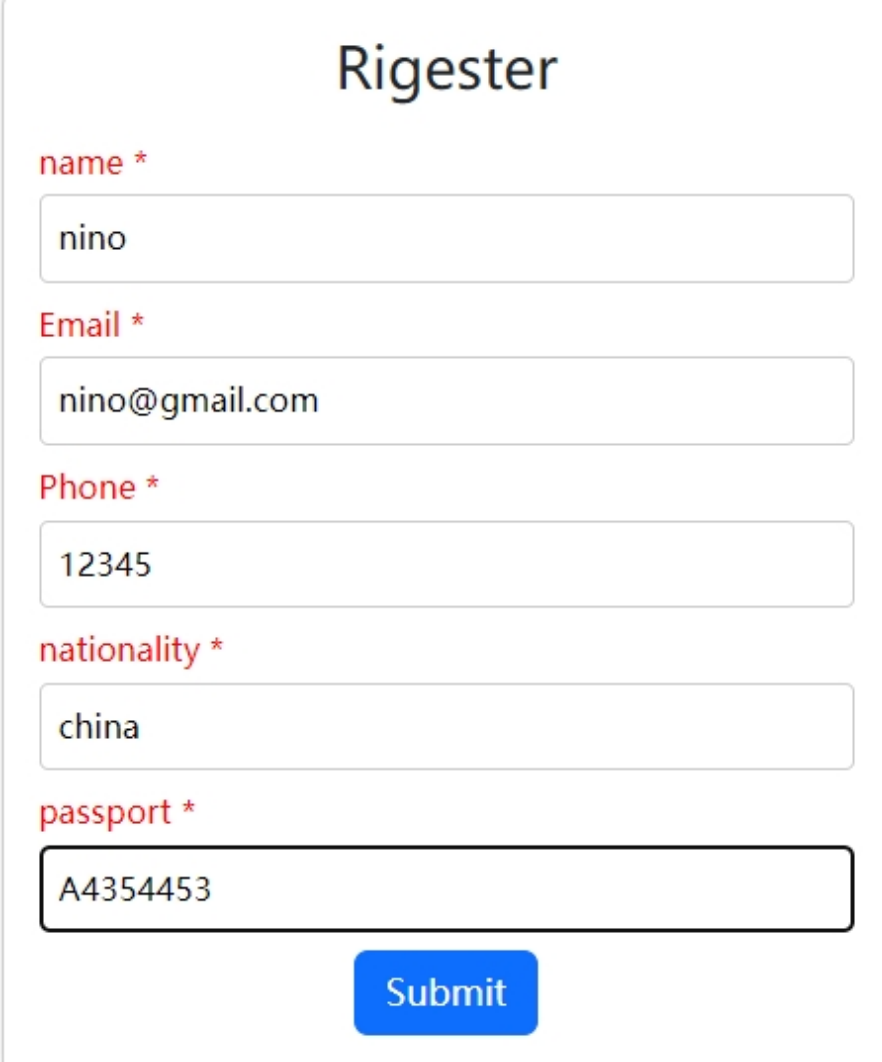
### 3.7.1 Applicable Personnel

- Tenant

### 3.7.2 Preconditions

- The Tenant has already connected his account via MetaMask.
- The landlord has registered the personal information and activated this contract

### 3.7.3 Description



**Rigester**

**name \***

nino

**Email \***

nino@gmail.com

**Phone \***

12345

**nationality \***

china

**passport \***

A4354453

**Submit**

**Figure 3.7:** The image of tenant registration

the tenant fills in the relevant information including name, email, phone, nationality, passport to register their information in the smart contract.

### 3.7.4 Postconditions

- - If the tenant successfully registers the information, the browser link redirects to the tenant home page.
- - If the tenant fails, there may be a problem with the information filled in, and you need to re-register according to the prompt.

### 3.7.5 Main Flow

- **step 1:** Fill in each information as required.
- **step 2:** click the Submit button.
- **step 3:** Agree to the request on MetaMask.

### 3.7.6 Frontend description

After clicking the submit button, the front end will judge whether the user has filled in all the information, and at the same time will judge that the information conforms to the corresponding format, if one of them does not meet, it can not submit and will display the corresponding promotion information.

### 3.7.7 Interfaces and Interactions:

- **addTenant():** Automatically evaluated gas. Accessible by any users. Need to pass any parameters of 'tenantAddress', 'createTime', 'name', 'email', 'phone', 'nationality', and 'passport'. Register basic personal information to the smart contract.

### 3.7.8 Security Analysis

This information is immutable and only viewable by the individual, ensuring privacy security.

## 3.8 Tenant Dashboard

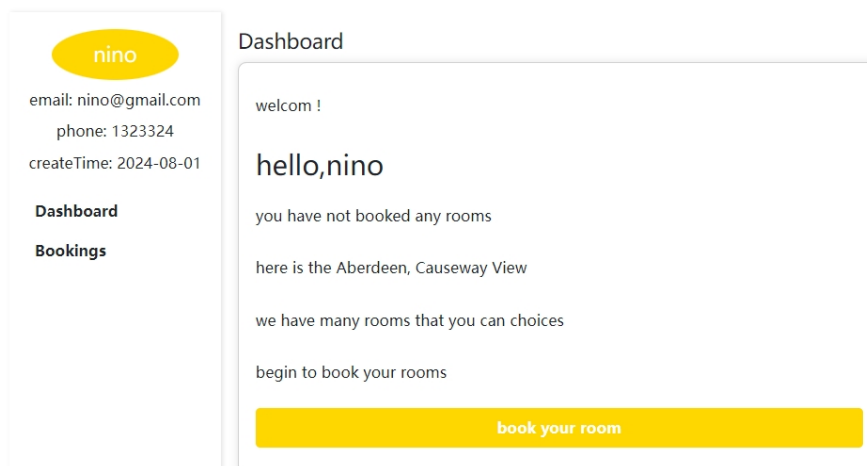
### 3.8.1 Applicable Personnel

- Tenant

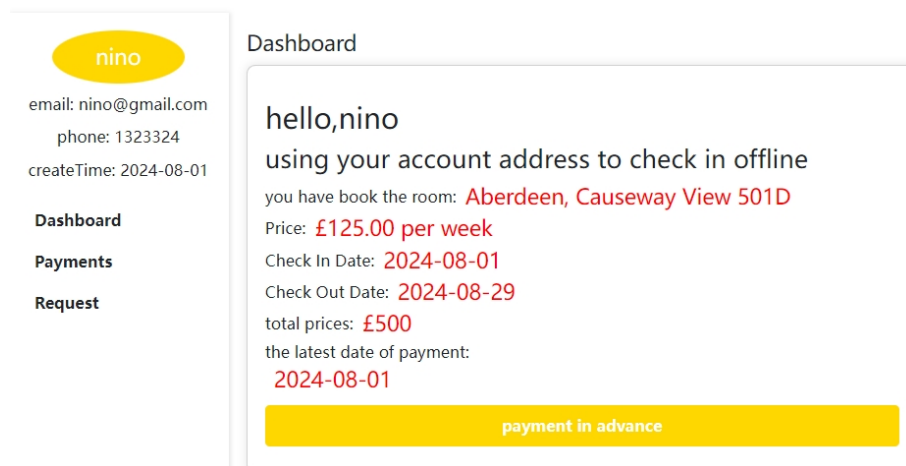
### 3.8.2 Preconditions

- The Tenant has already connected his account via MetaMask.
- The Tenant has registered the personal information.

### 3.8.3 Description



**Figure 3.8:** The tenant dashboard of not renting any room



**Figure 3.9:** The tenant dashboard of renting one room

the tenant can view the personal information. In addition, if you have not rented any room, you can click the book your room button to the room page. If you have already rented one room, you will get the detailed information of the tenancy, if you have pay the all cost, will show that you have pay all cost, but if you have not pay all the cost, will show the payment in advance button in the bottom of the box.

### 3.8.4 Postconditions

- - If Click the book the room button, switch to the booking page.
- - If click the payment in advance button, will switch to the payments page.

### 3.8.5 Main Flow

- **step 1:** open the system website link, the front-end switches to the dashboard page based on the corresponding logic.
- **step 2:** If you have already rented one room, you can check the detailed tenancy information.

- **step 3:** If you have already rented one room, you can click the payment in advance button to the payment page.
- **step 4:** if you have not rented any room, click the book the room button to the booking page.

### 3.8.6 Frontend description

The front end will display different page layouts according to the information returned by the interface. In addition, since the time returned by the interface is a timestamp, the front end also needs to write the corresponding algorithm to convert the timestamp into yyyy-mm-dd character display.

### 3.8.7 Interfaces and Interactions:

- **hasRegister():** No gas required. Accessible by any users. Need to pass parameters of address(your Ethereum account address). Determine whether the information has been registered by the address, if registered, the corresponding personal information is returned.
- **hasRentInfo():** No gas required. Accessible by any users. Need to pass parameters of address(your Ethereum account address) and curTime. Determine whether you have rented one room by your account address and current time, if you have rented one room, then return the corresponding tenancy information, if not return empty.

### 3.8.8 Security Analysis

This information is immutable and only viewable by the individual, ensuring privacy security.

## 3.9 Booking Room

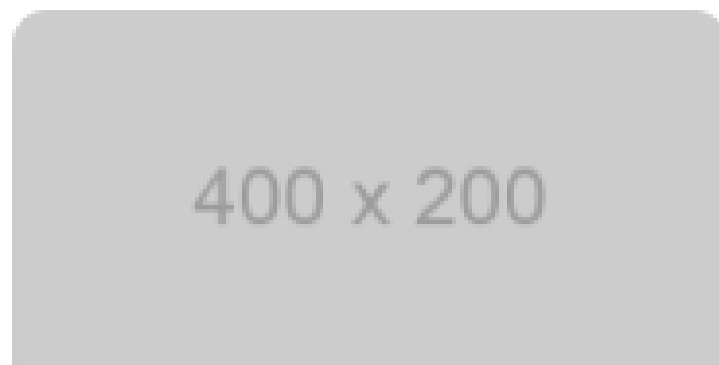
### 3.9.1 Applicable Personnel

- Tenant

### 3.9.2 Preconditions

- The Tenant has already connected his account via MetaMask.
- The Tenant has registered the personal information
- At least one room available

### 3.9.3 Description



**Address:** Aberdeen,  
Causeway View

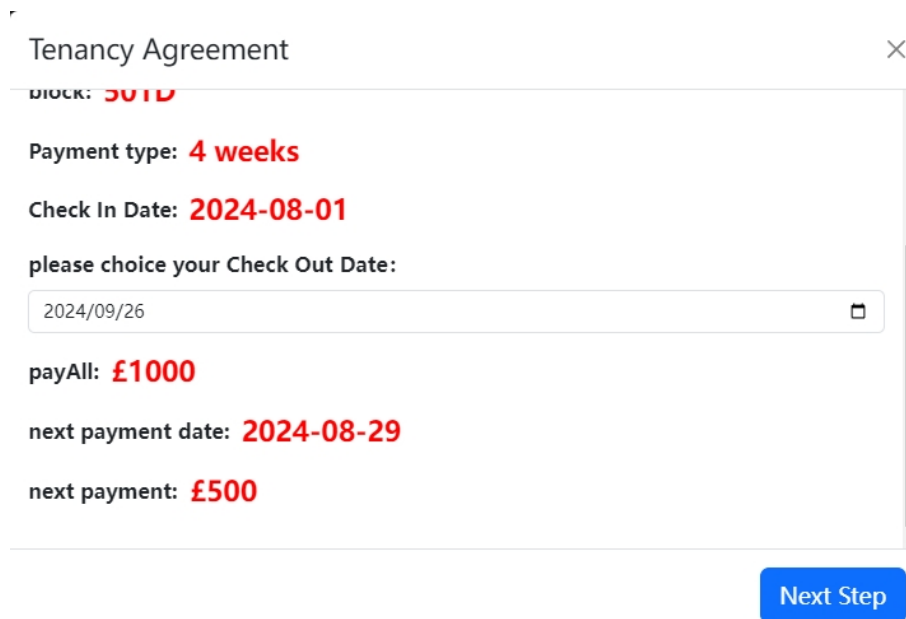
**Block:**501D

**Room type:**Single

**Price:** £125.00 per  
week

**book the room**

Figure 3.10: booking room



Tenancy Agreement

Block: 301D

Payment type: 4 weeks

Check In Date: 2024-08-01

please choice your Check Out Date:

2024/09/26

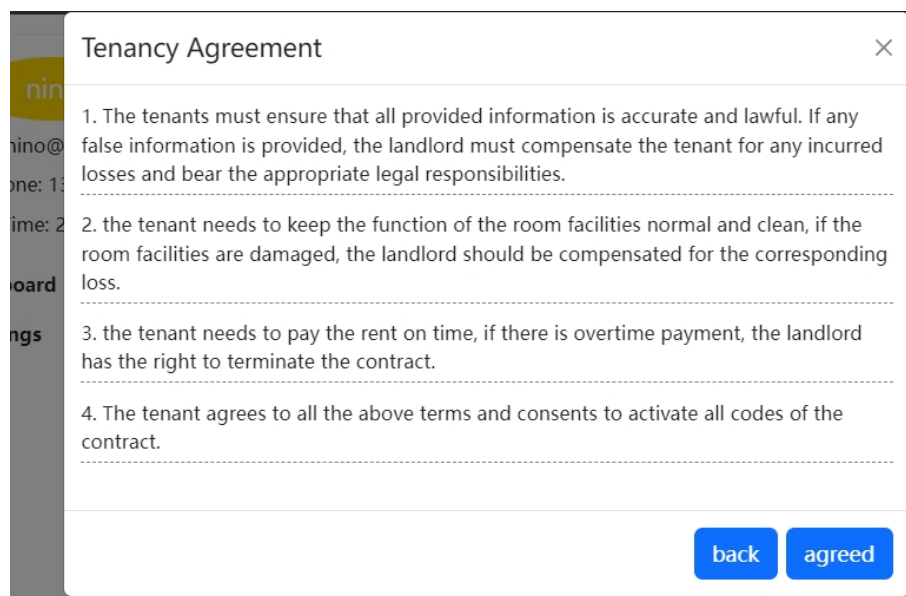
payAll: £1000

next payment date: 2024-08-29

next payment: £500

Next Step

**Figure 3.11:** the first step of booking room



Tenancy Agreement

1. The tenants must ensure that all provided information is accurate and lawful. If any false information is provided, the landlord must compensate the tenant for any incurred losses and bear the appropriate legal responsibilities.
2. the tenant needs to keep the function of the room facilities normal and clean, if the room facilities are damaged, the landlord should be compensated for the corresponding loss.
3. the tenant needs to pay the rent on time, if there is overtime payment, the landlord has the right to terminate the contract.
4. The tenant agrees to all the above terms and consents to activate all codes of the contract.

back agreed

**Figure 3.12:** the second step of booking room

The tenant can view available rooms, choose the room they want to rent, and click the booking button. The front end will automatically use the current date as the rental start date. The tenant can choose the end date, and the front end will adjust the total fee based on the chosen end date. After clicking next, the tenant can view the relevant contract terms. After carefully reading the contract terms, if the tenant agrees to all terms, they can click confirm to sign the rental contract.

### 3.9.4 Postconditions

- - If you rent one room successfully, you will be redirected to the home page.
- - If you fail, will display the error message for you, you need try again.

### 3.9.5 Main Flow

- **step 1:** Choice one room you want to rent, and click the book the room button.
- **step 2:** choice your check out date in the pop-up box and click the Next step button.
- **step 3:** Go through the Tenancy agreement terms, and click the agreed button.
- **step 4:** Agree to the request on MetaMasK.

### 3.9.6 Frontend description

The front end use the react-bootstrap/Modal as the pop-up box. the front end will generate the current time as the contract start time using JavaScript. Then, it will guide the tenant to choose the rental end time and adjust the total fee based on the chosen end time. The front end make sure that tenant's chosen time interval is 28 days.

### 3.9.7 Interfaces and Interactions:

- **getAllRooms():** No gas required. Accessible by any users. Not need to any parameters. Get all rooms.
- **getAgreementTermsforuser():** No gas required. Accessible by any tenant. Not need to any parameters. Get the tenant Agreement.
- **getAgreement():** Automatically evaluated gas. Accessible by any tenants. Need to pass parameters of parameters 'tenantId', 'roomId', 'createTime', 'endTime', 'nextTime', 'nextPay', 'payTotal', and

### 3.9.8 Security Analysis

the function is a core function. After the tenant chooses their desired room, the front end automatically sets the current time as the contract start time to ensure the contract's effectiveness. The end time interval is set to 28 days to ensure uniformity. All visible rooms are real and valid, and cannot be modified by anyone, ensuring the authenticity of the information and solving the trust issues of traditional contracts. The contract terms viewed by the tenant are also immutable. All these aspects ensure the transparency and efficiency of the rental process, providing higher security for the tenant.

## 3.10 Tenant Payment

### 3.10.1 Applicable Personnel

- Tenant

### 3.10.2 Preconditions

- The Tenant has already connected his account via MetaMask.
- The Tenant has registered the personal information
- the Tenant has rented a room but has not fully paid the rent.
- There is a sufficient balance in the tenant account

### 3.10.3 Description

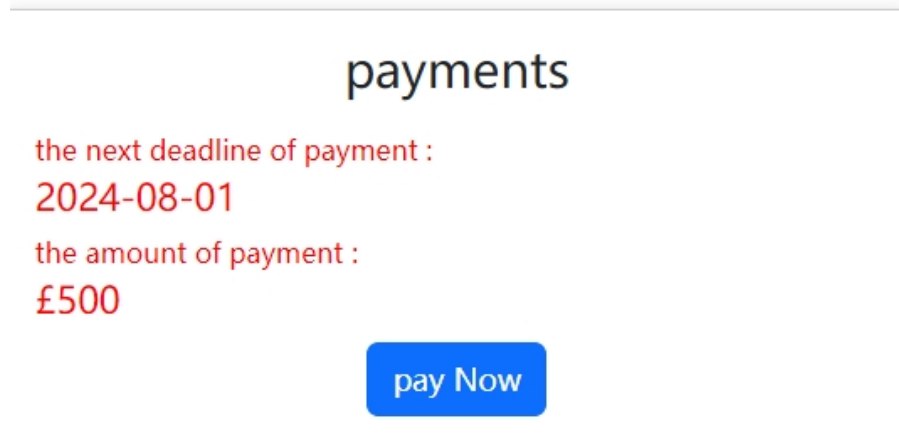


Figure 3.13: payment

The tenant needs to click the payment button to pay one month's rent before the 'nextTime' attribute in the rental information. After the tenant confirms the payment, the system will automatically update the next payment time.

### 3.10.4 Postconditions

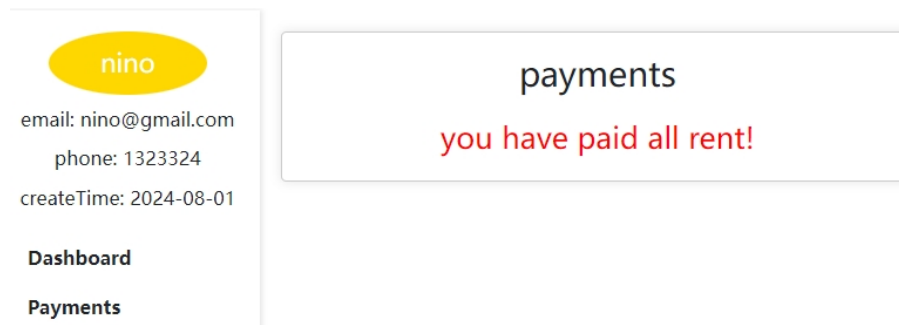


Figure 3.14: paid all payment

- - If after you click the payNow button, the next deadline of payment is less than the date of check out, the next deadline of payment will change.
- - If after you click the payNow button, the next deadline of payment is Greater than or equal to the date of check out, the page will show that you have pay the all cost.

### 3.10.5 Main Flow

- **step 1:** Click the payment tab to the payment page.
- **step 2:** Click pay Now button .
- **step 3:** Agree to the request on MetaMasK.

### 3.10.6 Frontend description

The front end use javascript method to modify the timestamp. display different page based on the data returned from interfaces.



### 3.10.7 Interfaces and Interactions:

- `payRent()`: Automatically evaluated gas. Accessible by any tenants. Need to pass parameters of 'uint256 rentId' and 'uint256 nextTime'. Automatically pay one month's rent and update the 'nextTime' value and the next payment time.

### 3.10.8 Security Analysis

The payment process is separate from signing the rental contract. If the tenant signs the rental contract but does not pay according to the contract, the contract will immediately become invalid. These processes are automated to ensure both parties' rights and improve the rental market's efficiency and transparency. Moreover, the tenant does not need to perform any extra calculations as the payment time and amount are automatically calculated by the contract, increasing efficiency. The entire process is anonymous, with payment made via Ether, ensuring both parties' privacy.

## 3.11 Submit Request

### 3.11.1 Applicable Personnel

- Tenant

### 3.11.2 Preconditions

- The Tenant has already connected his account via MetaMask.
- The Tenant has registered the personal information.
- the tenant has rented a room that has not yet expired.

### 3.11.3 Description

nino

email: nino@gmail.com

phone: 1323324

createTime: 2024-08-01

Dashboard

Payments

Request

Category

category

Enter full description

give you descriptions

Submit

Figure 3.15: submit request

If there are any issues during the rental period, the tenant can submit a request to describe their problems and wait for the landlord to resolve them.

#### **3.11.4 Postconditions**

- - after you click the submit successfully, an successful pop-up box will be display, the page will remain, you can submit again.
- - after you click the submit button unsuccessfully, the error messages will be displayed.

#### **3.11.5 Main Flow**

- **step 1:** Click the request tab on the front-end page.
- **step 2:** fill in the 'Category' and 'Description' of request and click the submit button.

#### **3.11.6 Frontend description**

The front end will judge if the category and description is correct.

#### **3.11.7 Interfaces and Interactions:**

- `createRequest()`: Automatically evaluated gas. Accessible by tenants. Need to pass parameters 'roomInfo', 'TenantId', 'requestType', 'description', and 'createTime'. Create a request to the smart contract.

#### **3.11.8 Security Analysis**

Only room information and related descriptions are provided, without any personal information, ensuring privacy. The request is recorded in the contract and cannot be changed. If the request is not addressed, or if the landlord does not fulfill the contract terms, the tenant can use this request information to protect their rights. This security mechanism avoids future disputes.

## Chapter 4

# System Deployment

## 4.1 System Deployment

### 4.1.1 Deployment Environment Configuration

- **Ganache:** you need to install Ganache. You can download Ganache from the Truffle Suite website (<https://trufflesuite.com/ganache/>), Download and install the version for your operating system (Windows, macOS, Linux).
- **MetaMask:** MetaMask is a browser extension wallet that supports Chrome, Firefox, Brave, and Edge browsers. Visit [MetaMask official website](<https://metamask.io/>) and select an extension based on your browser and install it.
- **Node.js (npm):** npm is integrated in node.js, so you just access the official Node.js website. Select the Node.js version for your operating system and download the LTS (Long Term Support) version.
- **truffle:** Enter the command `npm install -g truffle` to install the global truffle package

### 4.1.2 Deployment Steps

- **step1:** - Open the Ganache application, You will see an interface that displays the details of the local blockchain, including accounts, balances, blocks, and more. By default, Ganache provides 10 pre-configured accounts, each with 100 Ether for testing, Here you can use the first address as your landlord's address. Ganache will act as a local Ethereum blockchain environment network
- **step2:** Get the code project from the Github, the project github address is <https://github.com/shunino/tenancy> application.git.

```

string[] private lordAgreementTerms;
function getAgreementTermsforlord() public returns (string[] memory) {
    // 先清空数组
    delete lordAgreementTerms;
    lordAgreementTerms.push("1. The landlord must update daily tasks e
    lordAgreementTerms.push("2. The landlord must ensure that all prov
    lordAgreementTerms.push("3. The landlord must resolve any reasonab
    lordAgreementTerms.push("4. If the landlord delays or neglects to
    lordAgreementTerms.push("5. The landlord agrees to all the above t
    return lordAgreementTerms;
}

string[] private userAgreementTerms;
function getAgreementTermsforuser() public returns (string[] memory) {
    // 先清空数组
    delete userAgreementTerms;
    userAgreementTerms.push("1. The tenants must ensure that all provi
    userAgreementTerms.push("2. the tenant needs to keep the function
    userAgreementTerms.push("3. the tenant needs to pay the rent on tim
    userAgreementTerms.push("4. The tenant agrees to all the above ter
    return userAgreementTerms;
}

```

Figure 4.1: the agreement terms

- **step3:** Access the tenancy-application/truffle/contracts/TenancyContract.sol file, You can modify the contract terms in the getAgreementTermsforlord and getAgreementTermsforuser methods according to your own needs.

```

1 |const TenancyContract = artifacts.require("TenancyContract");
2
3 |module.exports = function(deployer, network, accounts) {
4 |    // deployAccount is the address of the landlord
5 |    //const deployAccount = accounts[0];
6 |    const deployAccount = '0x4B32AE490B3e7904688b0c0A09dFD67794cE2349';
7 |
8 |    deployer.deploy(TenancyContract, { from: deployAccount });
9 |};

```

Figure 4.2: the deployment address

- **step4:** Accessing the tenancy-application/truffle/migrations/1\_deploy\_contracts.js file, you need to change the address here to your own landlord's address, which will be used for deployment.

```
truffle-config.js
61 networks: {
62   // Useful for testing.
63   // if it's defined here
64   // You should run a client
65   // tab if you use this
66   // options below to some
67   //
68   development: {
69     host: "127.0.0.1",
70     port: 7545,
71     network_id: "*",
72   },
73   //
```

### Figure 4.3: connect the local ganache

- **step5:** Modify the `truffle-config.js` file to connect your project to your local ganache network (based on your ganache network information).

```

User@LAPTOP-C195VSP0 MINGW64 /d/smart-contract/tenancy-application/truffle (main)
$ truffle migrate --reset

Compiling your contracts...
=====
> Compiling .\contracts\TenancyContract.sol
> Compiling .\contracts\TenancyContract.sol
> Artifacts written to D:\smart-contract\tenancy-application\client\src\contracts
> Compiled successfully using:
   - solc: 0.8.18+commit.87f61d96.Emscripten.clang

Starting migrations...
=====
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)

1_deploy_contracts.js
=====

  Replacing 'TenancyContract'
  -----
  > transaction hash: 0x118b26e4400ad1c3ff3ce82e23d70a2e31bfa683854063d0a80cf1f5c6aa9d71
- Blocks: 0          Seconds: 0
  > Blocks: 0          Seconds: 0
  > contract address: 0x4DA69A7152e08bD3E62063c990F55Bb50820bBB8
  > block number:      248
  > block timestamp:    1722545849
  > account:           0x4B32AE490B3e7904688b0c0A09dFD67794cE2349
  > balance:            174.214089327925369692
  > gas used:           4883282 (0x4a8352)
  > gas price:          2.500000008 gwei
  > value sent:         0 ETH
  > total cost:         0.012208205039066256 ETH

  > Saving artifacts
  -----
  > Total cost:        0.012208205039066256 ETH

Summary
=====
> Total deployments: 1

```

Figure 4.4: *deploy<sub>successful</sub>*

- **step6:** Run the command `truffle migrate` in the `tenancy-application` directory To deploy this contract to the front-end project location `tenancy-application.json`. Your contract project has now been successfully deployed.
- **step7:** In the `tenancy-application/client` file directory, run the `npm install` command to install the front-end package required by the front-end project. Run `npm start` to launch the front-end project, typically at `http://localhost:8080/`.

## Chapter 5

# System Security Analysis

### 5.1 Information Reliability Analysis

Information reliability is crucial in maintaining the integrity and trustworthiness of the smart rental contract system. This section evaluates the mechanisms and protocols in place to ensure that the information processed and stored within the system is accurate and reliable.

#### 5.1.1 Data Validation and Verification

- **Input Validation:** Ensure that all data entered into the system is validated to prevent invalid or malicious data from being processed. Techniques such as input sanitization and validation rules are implemented.
- **Transaction Verification:** Each transaction on the blockchain is verified by network nodes, ensuring that only valid transactions are recorded. This reduces the risk of fraudulent or erroneous data entries.

#### 5.1.2 Redundancy and Consistency

- **Data Redundancy:** Employ data redundancy techniques to store multiple copies of critical data. This ensures that in case of data corruption or loss, a reliable backup is available.
- **Consistency Checks:** Regular consistency checks are conducted to compare different copies of data and ensure they are identical. Discrepancies are flagged for investigation.

#### 5.1.3 Trust Mechanisms

- **Consensus Protocols:** Use consensus protocols like Proof of Work (PoW) or Proof of Stake (PoS) to achieve agreement on the blockchain state among distributed nodes. This ensures that all nodes share the same reliable information.
- **Reputation Systems:** Implement reputation systems to rate the reliability of nodes and participants in the network. Nodes with a history of reliable behavior are given more trust in the consensus process.

#### 5.1.4 Results and Discussion

The information reliability analysis indicates that the smart rental contract system employs robust mechanisms to ensure data accuracy and trustworthiness. However, continuous monitoring and updates to validation rules are necessary to adapt to new types of threats.

## 5.2 Information Security Analysis

Information security is paramount in protecting the smart rental contract system from unauthorized access, data breaches, and cyber-attacks. This section assesses the security measures implemented to safeguard information within the system.

### 5.2.1 Encryption and Data Protection

- **Data Encryption:** Sensitive data is encrypted both at rest and in transit using strong cryptographic algorithms such as AES-256. This ensures that even if data is intercepted, it cannot be read without the decryption key.
- **Secure Communication:** All communications between users and the system are secured using Transport Layer Security (TLS) protocols to prevent eavesdropping and man-in-the-middle attacks.

### 5.2.2 Access Control

- **Authentication:** Multi-factor authentication (MFA) is required for accessing the system, providing an additional layer of security beyond passwords.
- **Authorization:** Role-based access control (RBAC) is used to ensure that users have access only to the information and functions necessary for their role, reducing the risk of insider threats.

### 5.2.3 Monitoring and Incident Response

- **Security Monitoring:** Continuous monitoring of system activity through security information and event management (SIEM) systems helps in detecting anomalies and potential security incidents in real time.
- **Incident Response:** A well-defined incident response plan is in place, enabling quick identification, containment, and remediation of security breaches.

### 5.2.4 Results and Discussion

The information security analysis reveals that the smart rental contract system has implemented comprehensive security measures. However, regular security audits and updates to the security infrastructure are recommended to stay ahead of evolving threats.

## 5.3 Information Immutable Analysis

Immutability is a key feature of blockchain technology, ensuring that once information is recorded, it cannot be altered or deleted. This section examines how the smart rental contract system leverages blockchain immutability to enhance data security and integrity.

### 5.3.1 Blockchain Immutability

- **Immutable Ledger:** The blockchain maintains an immutable ledger of all transactions, ensuring that once a transaction is confirmed, it cannot be changed. This provides a verifiable history of all actions taken within the system.
- **Tamper-Evident Records:** Any attempt to alter past records would be immediately evident, as it would require changing all subsequent blocks, which is computationally infeasible in a well-secured blockchain network.



### 5.3.2 Cryptographic Hashing

- **Hash Functions:** Each block in the blockchain contains a cryptographic hash of the previous block, linking them together securely. This chain structure ensures that any modification to a block would invalidate all subsequent hashes.
- **Merkle Trees:** Merkle trees are used to efficiently and securely verify the integrity of data. Each leaf node of the tree is a hash of a data block, and non-leaf nodes are hashes of their respective child nodes, allowing quick and reliable verification of data integrity.

### 5.3.3 Smart Contract Immutability

- **Immutable Code:** Once deployed, smart contracts on the blockchain are immutable. This ensures that the contract's logic cannot be altered, providing certainty and trust in the automated execution of agreements.
- **Auditability:** The immutability of smart contracts allows for complete audit trails

## Chapter 6

# Challenges and Future Outlook

## 6.1 Challenges

### 6.1.1 Technical Challenges

- **Smart Contract Vulnerabilities:** One of the main challenges of smart contracts is their immutable nature once deployed on the blockchain. Any vulnerabilities or bugs in the code can lead to significant security issues, as evidenced by the infamous DAO attack. Ensuring the security and reliability of smart contracts is a critical technical challenge.
- **Blockchain Performance:** Current blockchain technologies, such as Ethereum, face performance bottlenecks when processing a high volume of transactions. High transaction fees and slow processing speeds can negatively impact the user experience and scalability of the smart rental contract system.
- **Data Privacy:** While blockchain offers transparency and immutability, this transparency can also lead to potential privacy issues since all transactions are publicly accessible. Balancing transparency with privacy protection is a key challenge that needs to be addressed.

### 6.1.2 Operational Challenges

- **User Adoption:** Despite the clear advantages of smart rental contracts, there may be resistance from users accustomed to traditional rental markets. Educating and encouraging users to understand and embrace this new technology requires significant effort and time.
- **Legal and Regulatory Issues:** The legal status and regulatory framework for smart contracts are still evolving. Different countries have varying legal requirements and regulations for blockchain and smart contracts, which could hinder cross-border applications of the system.

### 6.1.3 Market Challenges

- **Market Competition:** With the rapid development of blockchain technology, many companies are entering the smart rental contract market. Standing out in a competitive market and attracting more users is a significant challenge.

- **Trust Building:** Although blockchain technology reduces the need for intermediaries, a fully decentralized system still requires user trust. Building this trust through technological reliability and quality service is crucial.

## 6.2 Future Outlook

### 6.2.1 Technological Advancements

- **Smart Contract Auditing:** Future developments should focus on advanced smart contract auditing tools and methodologies to ensure security and reliability. Techniques such as formal verification can be used to detect and fix potential vulnerabilities before deployment.
- **Layer 2 Solutions:** Implementing Layer 2 solutions (such as Lightning Network and sidechains) can enhance blockchain's transaction processing capabilities and reduce transaction fees, thereby improving system scalability and user experience.
- **Privacy-Enhancing Technologies:** Research and application of privacy-enhancing technologies like zero-knowledge proofs (zk-SNARKs) can ensure transaction transparency while protecting user privacy.

### 6.2.2 Operational Improvements

- **User Education and Outreach:** Enhancing user education through training and promotional efforts can help users understand the benefits and usage of smart rental contract systems, increasing adoption rates.
- **Standardization and Compliance:** Collaborating with legal experts and regulatory bodies to promote the standardization and compliance of smart contracts will ensure that the system meets the legal and regulatory requirements of different countries and regions.

### 6.2.3 Market Strategies

- **Differentiation Strategy:** Creating a competitive edge by offering unique features and superior user experiences, such as customized rental services, enhanced security measures, and excellent customer support.
- **Ecosystem Development:** Building and expanding the smart rental contract ecosystem by attracting more developers and partners to collaborate on technological advancements and market expansion. Partnerships with other blockchain projects and traditional enterprises can lead to resource sharing and mutual benefits.

## Chapter 7

# Conclusions

In conclusion, the implementation and adoption of smart rental contract systems represent a significant advancement in the rental market, leveraging the transformative power of blockchain technology. This paper has explored the multifaceted aspects of these systems, including their benefits, challenges, and future outlook.

Smart rental contracts offer substantial benefits such as enhanced security, transparency, and efficiency. By automating the execution and enforcement of contract terms, these systems reduce the need for intermediaries, minimize disputes, and streamline the rental process. The immutable and transparent nature of blockchain technology ensures that all transactions are securely recorded and easily auditable, fostering trust among all parties involved.

However, several challenges must be addressed to fully realize the potential of smart rental contracts. Technical challenges include ensuring the security and reliability of smart contracts, addressing blockchain performance bottlenecks, and protecting user data privacy. Operational challenges encompass user adoption, legal and regulatory compliance, and building user trust in a decentralized system. Market challenges involve differentiating the system in a competitive landscape and fostering a robust ecosystem of developers and partners.

Looking ahead, the future of smart rental contracts is promising, with potential advancements in smart contract auditing, Layer 2 solutions for scalability, and privacy-enhancing technologies. Operational improvements through user education and outreach, standardization, and compliance efforts will be crucial in driving adoption. Market strategies focusing on differentiation and ecosystem development will help in establishing a competitive edge and fostering innovation.

Through continuous technological innovation, comprehensive user education, and strict compliance management, smart rental contract systems can significantly enhance security, usability, and user experience. As blockchain technology matures and gains wider acceptance, these systems will play an increasingly important role in improving market efficiency, reducing costs, and enhancing transparency. Ultimately, this will provide users with safer, more convenient, and trustworthy rental services, supporting the digital transformation of the rental market on a global scale.

By addressing the identified challenges and leveraging future advancements, smart rental

---

contract systems have the potential to revolutionize the rental industry, offering a more efficient and secure alternative to traditional rental practices. The journey towards widespread adoption and integration of these systems will undoubtedly be complex, but the benefits they offer make it a worthwhile endeavor.

## **Bibliography**