

# Design and Implementation of Blockchain-based Smart Contract System for Tenancy

*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 housing tenancy market. Traditional tenancy methods face numerous security issues, such as information asymmetry, data leakage, and contract tampering. Additionally, tenancy transactions typically rely on intermediary agencies, which can lead to trust issues. Smart tenancy contracts manage property agreements automatically, ensuring transparency and security, and protecting the confidentiality of personal information and the immutability of contracts. This paper details the design and implementation of blockchain-based smart contract system for tenancy, highlighting the integration of frontend technologies for seamless interaction with smart contracts, 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 the housing tenancy market's 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	knowledge acquisition . . . . .	11
2.2	Smart Contracts . . . . .	12
2.2.1	Smart Contract Life Cycle . . . . .	12
2.2.2	knowledge acquisition . . . . .	12
2.3	Solidity . . . . .	13
2.3.1	Characteristisc of Solidity . . . . .	13
2.3.2	knowledge acquisition . . . . .	13
2.4	Ganache . . . . .	13
2.4.1	Characteristics of Ganache . . . . .	13
2.4.2	Knowledge Acquisition . . . . .	14
2.5	MetaMask . . . . .	14
2.5.1	Characteristics of MetaMask . . . . .	14
2.5.2	Knowledge Acquisition . . . . .	14
2.6	React.js . . . . .	15
2.6.1	Characteristics of React.js . . . . .	15
2.6.2	Knowledge Acquisition . . . . .	15
2.7	Web3.js . . . . .	15
2.7.1	Characteristics of Web3.js . . . . .	15
2.7.2	Knowledge Acquisition . . . . .	16
2.8	Truffle . . . . .	16
2.8.1	Characteristics of Truffle . . . . .	16
2.8.2	Knowledge Acquisition . . . . .	16
<b>3</b>	<b>System Functional Module Description</b>	<b>17</b>
3.1	Connecting the MetMask . . . . .	17
3.1.1	Applicable Personnel . . . . .	17
3.1.2	Preconditions . . . . .	17

3.1.3	Description . . . . .	17
3.1.4	Postconditions . . . . .	17
3.1.5	Main Flow . . . . .	17
3.1.6	Frontend description . . . . .	17
3.1.7	Interfaces and Interactions: . . . . .	18
3.1.8	Security Analysis . . . . .	18
3.2	Contract Activation . . . . .	18
3.2.1	Applicable Personnel . . . . .	18
3.2.2	Preconditions . . . . .	18
3.2.3	Description . . . . .	18
3.2.4	Postconditions . . . . .	19
3.2.5	Main Flow . . . . .	19
3.2.6	Frontend description . . . . .	19
3.2.7	Interfaces and Interactions: . . . . .	19
3.2.8	Security Analysis . . . . .	19
3.3	Landlord Dashboard . . . . .	19
3.3.1	Applicable Personnel . . . . .	19
3.3.2	Preconditions . . . . .	20
3.3.3	Description . . . . .	20
3.3.4	Postconditions . . . . .	20
3.3.5	Main Flow . . . . .	20
3.3.6	Frontend description . . . . .	20
3.3.7	Interfaces and Interactions: . . . . .	21
3.3.8	Security Analysis . . . . .	21
3.4	Add and Modify Room Status . . . . .	21
3.4.1	Applicable Personnel . . . . .	21
3.4.2	Preconditions . . . . .	21
3.4.3	Description . . . . .	22
3.4.4	Postconditions . . . . .	22
3.4.5	Main Flow . . . . .	22
3.4.6	Frontend description . . . . .	23
3.4.7	Interfaces and Interactions: . . . . .	23
3.4.8	Security Analysis . . . . .	23
3.5	View and Modify Tenant Request Lists . . . . .	23
3.5.1	Applicable Personnel . . . . .	23
3.5.2	Preconditions . . . . .	23
3.5.3	Description . . . . .	23
3.5.4	Postconditions . . . . .	24
3.5.5	Main Flow . . . . .	24
3.5.6	Frontend description . . . . .	24
3.5.7	Interfaces and Interactions: . . . . .	24
3.5.8	Security Analysis . . . . .	24

3.6	View Rental Information by Address . . . . .	24
3.6.1	Applicable Personnel . . . . .	24
3.6.2	Preconditions . . . . .	24
3.6.3	Description . . . . .	25
3.6.4	Postconditions . . . . .	25
3.6.5	Main Flow . . . . .	25
3.6.6	Frontend description . . . . .	25
3.6.7	Interfaces and Interactions: . . . . .	25
3.6.8	Security Analysis . . . . .	25
3.7	Tenant Registration . . . . .	26
3.7.1	Applicable Personnel . . . . .	26
3.7.2	Preconditions . . . . .	26
3.7.3	Description . . . . .	26
3.7.4	Postconditions . . . . .	27
3.7.5	Main Flow . . . . .	27
3.7.6	Frontend description . . . . .	27
3.7.7	Interfaces and Interactions: . . . . .	27
3.7.8	Security Analysis . . . . .	27
3.8	Tenant Dashboard . . . . .	27
3.8.1	Applicable Personnel . . . . .	27
3.8.2	Preconditions . . . . .	27
3.8.3	Description . . . . .	28
3.8.4	Postconditions . . . . .	28
3.8.5	Main Flow . . . . .	28
3.8.6	Frontend description . . . . .	29
3.8.7	Interfaces and Interactions: . . . . .	29
3.8.8	Security Analysis . . . . .	29
3.9	Booking Room . . . . .	29
3.9.1	Applicable Personnel . . . . .	29
3.9.2	Preconditions . . . . .	29
3.9.3	Description . . . . .	30
3.9.4	Postconditions . . . . .	31
3.9.5	Main Flow . . . . .	32
3.9.6	Frontend description . . . . .	32
3.9.7	Interfaces and Interactions: . . . . .	32
3.9.8	Security Analysis . . . . .	32
3.10	Tenant Payment . . . . .	32
3.10.1	Applicable Personnel . . . . .	32
3.10.2	Preconditions . . . . .	32
3.10.3	Description . . . . .	33
3.10.4	Postconditions . . . . .	33
3.10.5	Main Flow . . . . .	33

3.10.6	Frontend description . . . . .	33
3.10.7	Interfaces and Interactions: . . . . .	34
3.10.8	Security Analysis . . . . .	34
3.11	Submit Request . . . . .	34
3.11.1	Applicable Personnel . . . . .	34
3.11.2	Preconditions . . . . .	34
3.11.3	Description . . . . .	34
3.11.4	Postconditions . . . . .	35
3.11.5	Main Flow . . . . .	35
3.11.6	Frontend description . . . . .	35
3.11.7	Interfaces and Interactions: . . . . .	35
3.11.8	Security Analysis . . . . .	35
<b>4</b>	<b>System Deployment and Test guidance</b>	<b>36</b>
4.1	System Deployment . . . . .	36
4.1.1	Deployment Environment Configuration . . . . .	36
4.1.2	Deployment Steps . . . . .	36
4.2	System Test Guidance . . . . .	39
4.2.1	Detailed steps . . . . .	40
<b>5</b>	<b>System Security Analysis</b>	<b>41</b>
5.1	Information Reliability Analysis . . . . .	41
5.1.1	Data Validation and Verification . . . . .	41
5.1.2	Trust Mechanisms . . . . .	43
5.2	Information Security Analysis . . . . .	43
5.2.1	Encryption and Data Protection . . . . .	43
5.2.2	Access Control . . . . .	44
5.2.3	Event Response and recording . . . . .	44
5.3	Information Immutable Analysis . . . . .	45
5.3.1	Blockchain Immutability . . . . .	45
5.3.2	Cryptographic Hashing . . . . .	45
5.3.3	Smart Contract Immutability . . . . .	46
<b>6</b>	<b>Challenges and Future Outlook</b>	<b>47</b>
6.1	Challenges . . . . .	47
6.1.1	Technical Challenges . . . . .	47
6.1.2	Operational Challenges . . . . .	48
6.2	Future Outlook . . . . .	48
6.2.1	Technological Advancements . . . . .	48
6.2.2	Operational Improvements . . . . .	48
<b>7</b>	<b>Conclusions</b>	<b>49</b>



## Chapter 1

# Introduction

As the global housing tenancy market continues to grow, it has become a significant and vital economic sector worldwide. However, the traditional tenancy market faces numerous challenges, including information asymmetry, difficulties in contract enforcement, high transaction costs, and frequent disputes [9]. Tenancy 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 tenancy 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 [11]. Within the blockchain context, smart contracts are scripts stored on the blockchain. (They can be thought of as roughly analogous to stored procedures in relational database management systems) [3]. Smart contracts for tenancy can automate the execution of tenancy agreements, handle rent payments automatically, and effectively protect the rights of both parties, thereby enhancing the efficiency and transparency of the tenancy market and reducing the operational risks associated with intermediaries. Furthermore, blockchain's encryption technology and distributed storage ensure the security and privacy of tenancy transaction data, effectively preventing data tampering and unauthorized access, and reducing the risk of information leakage. Through these security mechanisms, blockchain-based smart contract system for tenancy 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 blockchain-based smart contract system for tenancy in the modern housing 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 tenancy market. This paper also explores how to use this tool more efficiently and maximize its security benefits.

We designed and implemented a blockchain-based smart contract system for tenancy. Through in-depth research on smart contracts, we aim to demonstrate their specific application scenarios and technical implementation processes in the tenancy market. We will explore how

smart contracts can improve overall market efficiency by automating tenancy agreement execution, ensuring transaction transparency, protecting user privacy, and reducing operational risks. Furthermore, we investigate the integration of frontend technologies to facilitate user interactions with smart contracts, ensuring both ease of use and robust security.

In summary, this research is not only an exploration and implementation of blockchain-based smart contract system for tenancy 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 tenancy market participants with a more secure, efficient, and transparent transaction method, thereby supporting a safer digital transformation of the tenancy market.

## 1.2 Outline

This dissertation is organized into 7 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 the different functional module interact to achieve the system's objectives.

Chapter 4: System Deployment - This chapter describes the software required for the deployment and the configuration and steps of the system deployment environment. It includes instructions on setting up the necessary blockchain infrastructure and integrating frontend technologies.

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 after developing the system. 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. Initially conceptualized for Bitcoin by an anonymous entity known as Satoshi Nakamoto in 2008, blockchain has since been applied to various domains beyond cryptocurrencies [12].

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 [12]. The decentralized nature of blockchain eliminates the need for a central authority, thereby reducing the risk of corruption and fraud.

### 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 [4].
- **Transparency:** All transactions on the blockchain are visible to all participants [12]. This transparency fosters trust among users, as they can independently verify and audit the transactions.
- **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 [13].
- **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 [1].

### 2.1.2 knowledge acquisition

Here is some knowledge to help me better complete the tenancy system.

- **Decentralization:** it ensures that tenancy agreements can be managed without the need for a central authority, such as a property management company or legal intermediary. This reduces costs and eliminates the potential for biased decision-making, as all transactions and agreements are directly recorded and enforced on the blockchain by the participating parties themselves.
- **Transparency:** it ensures that all parties involved in the tenancy system access to the same data. This visibility builds trust and reduces disputes since all actions (e.g., payment of rent, return of security deposits) are logged in a tamper-proof manner.
- **Immutability:** it provides security to both parties in the tenancy agreement. Once a tenancy is recorded on the blockchain, its terms cannot be altered, which protects against fraudulent claims or unauthorized changes to the contract.

## 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. 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].

### 2.2.1 Smart Contract Life Cycle

- **Create a contract:** After the parties negotiate the contents of the contract and reach an agreement. Lawyers draft agreements, which software engineers call smart contracts, including the logical connection between contracts. The creation of intelligent contracts requires multiple iterations [10].
- **Deployment contract:** Deploy contracts written by software engineers on the blockchain. All parties can access the contract content through blockchain [10].
- **Contract execution:** After the condition is met and triggered, the smart contract will be automatically executed, and the things and the updated state will be stored in the blockchain [10].
- **Completion of the contract:** Completion of the contract. After the intelligent contract is executed, the transaction information and status are stored in the blockchain [10].

### 2.2.2 knowledge acquisition

- **Smart Contract Lifecycle:** Understanding the lifecycle of smart contracts—from creation, deployment, and execution to completion—ensures that the tenancy system is well-structured and can automate and enforce lease agreements effectively.
- **Automation of Processes:** By applying the principle of contract execution, the system can automate essential processes like rent payments and lease terminations when predefined conditions are met, enhancing efficiency and reducing the need for manual intervention.

- **Transparency and Traceability:** The deployment of contracts on the blockchain ensures that all parties involved in the tenancy can access and verify the contract's terms and status at any time, fostering trust and reducing disputes.
- **Security and Immutability:** Security and Immutability: Once a contract is completed, the transaction details are securely stored on the blockchain, ensuring that the records are tamper-proof and trustworthy for all involved parties.

## 2.3 Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs that govern the behavior of accounts within the Ethereum state [2].

### 2.3.1 Characteristic of Solidity

- **Inheritance and Modularity:** Solidity supports inheritance and modular code structure, enabling developers to reuse code efficiently. This feature is especially useful when creating large and complex smart contracts [2].
- **Contract-oriented Language:** Solidity is designed specifically for writing contracts. Each contract in Solidity can contain its state variables, functions, and events, facilitating the creation of complex blockchain applications [2].
- **Libraries and Interfaces:** Solidity allows developers to create libraries that contain reusable code, as well as interfaces to define how different contracts interact with each other. This modularity supports the creation of scalable and maintainable systems [2].
- **Supports Custom Data Types:** Solidity enables developers to define complex custom data structures, enhancing the ability to model real-world scenarios within smart contracts [2].
- **Security Features:** Solidity includes features designed to enhance the security of smart contracts, such as access control modifiers and fallback functions, reducing the risk of vulnerabilities and attacks [2].

### 2.3.2 knowledge acquisition

Learned solidity syntax, how to create a smart contract on the Ethereum blockchain with solidity language. Developers use it to create contracts that handle anything from simple transactions to complex decentralized applications

## 2.4 Ganache

Ganache is a personal blockchain for Ethereum development that the developer can use to deploy contracts, develop applications, and run tests. It provides developers with a local blockchain network for testing and debugging smart contracts [8].

### 2.4.1 Characteristics of Ganache

- **User-Friendly Interface:** Ganache offers a graphical user interface (GUI) that allows developers to easily manage accounts, view transactions, and monitor events in real-time [8].

- **Local Blockchain Network:** Ganache simulates a blockchain network on the local machine, providing a fast and easy setup for testing smart contracts without the need for a public network [8].
- **Customizable Environment:** Developers can customize the blockchain environment, including gas price, block time, and account balances, to suit their testing needs [8].
- **Transaction Logging:** Ganache logs all transactions and events, making it easier to debug and analyze contract behavior during development [8].
- **Integrated with Truffle Suite:** Ganache is part of the Truffle Suite, which offers a suite of tools for Ethereum development, allowing seamless integration with other development tools such as Truffle and Drizzle [8].

#### 2.4.2 Knowledge Acquisition

In the Blockchain-based Smart Contract System for Tenancy, Ganache is used to create and manage a local blockchain network for testing and development. By using Ganache, developers can deploy and test smart contracts in a controlled environment, ensuring their functionality before deploying them to a public network.

## 2.5 MetaMask

@miscMetaMask, MetaMask is a browser extension that serves as a cryptocurrency wallet and gateway to blockchain applications. It allows users to manage their Ethereum accounts and interact with decentralized applications (dApps) [6].

#### 2.5.1 Characteristics of MetaMask

- **Wallet Functionality:** MetaMask allows users to store and manage Ethereum and ERC-20 tokens securely within their browser [6].
- **dApp Integration:** It enables users to interact with dApps directly from the browser, providing seamless access to blockchain applications [6].
- **Account Management:** Users can create and manage multiple Ethereum accounts, facilitating organization and separation of assets [6].
- **Network Configuration:** MetaMask supports multiple networks, allowing users to switch between the Ethereum mainnet, testnets, and custom networks [6].
- **Security Features:** MetaMask includes security measures such as password protection and seed phrases to ensure user data and assets remain secure [6].

#### 2.5.2 Knowledge Acquisition

In the Blockchain-based Smart Contract System for Tenancy, MetaMask is used to manage Ethereum accounts and interact with local blockchain network(ganache). By integrating MetaMask, users can securely connect to blockchain networks and execute transactions within applications.

## 2.6 React.js

React.js is a JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create dynamic and interactive web applications efficiently [15].

### 2.6.1 Characteristics of React.js

- **Component-Based Architecture:** React.js encourages the use of reusable components, making it easier to build and maintain complex UIs [15].
- **Virtual DOM:** React.js uses a virtual DOM to optimize rendering performance, updating only the parts of the UI that change [15].
- **One-Way Data Flow:** React enforces a unidirectional data flow, which helps maintain predictable data management and application state [15].
- **JSX Syntax:** React uses JSX, a syntax extension that allows developers to write HTML-like code within JavaScript, enhancing readability and ease of use [15].
- **Ecosystem and Community:** With a vast ecosystem of libraries and a strong community, React.js provides a wealth of resources for developers [15].

### 2.6.2 Knowledge Acquisition

In the Blockchain-based Smart Contract System for Tenancy, React.js is used to build interactive UIs for web applications. Developers leverage its component-based structure to create scalable and maintainable frontends [15].

## 2.7 Web3.js

Web3.js is a JavaScript library that allows developers to interact with the Ethereum blockchain. It provides APIs for sending transactions, interacting with smart contracts, and managing user accounts [5].

### 2.7.1 Characteristics of Web3.js

- **Ethereum Interaction:** Web3.js enables developers to communicate with the Ethereum blockchain, facilitating smart contract deployment and execution [5].
- **Account Management:** It provides tools to manage Ethereum accounts and handle transactions programmatically [5].
- **Event Listening:** Web3.js supports event subscriptions, allowing developers to listen for blockchain events and react to changes in contract state [5].
- **Compatibility:** Web3.js is compatible with various Ethereum nodes and services, including Infura and MetaMask [5].
- **Comprehensive API:** It offers a rich set of APIs for interacting with the Ethereum network, covering a wide range of functionalities [5].

### 2.7.2 Knowledge Acquisition

In the Blockchain-based Smart Contract System for Tenancy, Web3.js is used to interact with the Ethereum blockchain from web applications. Developers use it to send transactions, call contract methods, and listen for blockchain events.

## 2.8 Truffle

Truffle is a development environment, testing framework, and asset pipeline for Ethereum. It simplifies smart contract development by providing tools for compiling, deploying, and testing contracts [7].

### 2.8.1 Characteristics of Truffle

- **Smart Contract Lifecycle Management:** Truffle provides tools for compiling, deploying, and managing smart contracts throughout their lifecycle [7].
- **Automated Testing:** It includes a testing framework that supports automated smart contract testing using JavaScript and Solidity [7].
- **Migration System:** Truffle offers a migration system to manage contract deployment scripts and track changes across different environments [7].
- **Network Management:** Truffle supports configuration for multiple networks, enabling easy deployment to both local and public Ethereum networks [7].
- **Integration with Ganache:** Truffle integrates seamlessly with Ganache, providing a complete development suite for Ethereum applications [7].

### 2.8.2 Knowledge Acquisition

In the Blockchain-based Smart Contract System for Tenancy, Truffle is used to streamline the development process for Ethereum smart contracts. Developers use it for compiling, deploying, and testing contracts efficiently.



## Chapter 3

# System Functional Module Description

### 3.1 Connecting the MetMask

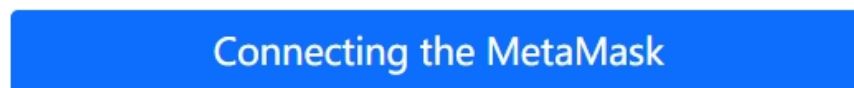
#### 3.1.1 Applicable Personnel

- landlord
- tenant

#### 3.1.2 Preconditions

- The host has deployed the system project
- browser has downloaded and installed the MetaMask plugin
- the MetaMask has be 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 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 users are prompted to re-connect the 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 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.

### 3.1.7 Interfaces and Interactions:

- **web3.eth.requestAccounts()**: web.js library method. No need to pass any parameters. Check whether metaMask is connected, if not, connecting to MetaMask, obtain the connected account information.

### 3.1.8 Security Analysis

only Ethereum accounts can access this site, and 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

The image shows a web form titled "Rigester to Activate". It contains the following fields and elements:

- name \***: A text input field containing "landlord".
- Email \***: A text input field containing "lanlord@gmail.com".
- Phone \***: A text input field containing "1234556".
- nationality \***: A dropdown menu with "UK" selected.
- passport \***: A text input field containing "A54545".
- \* agree with the agreement before you submit**: A checkbox that is checked.
- Submit**: A blue button.

**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 the landlord 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 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.

### 3.2.7 Interfaces and Interactions:

- **isActivate():** Self-evaluated gas. Accessible by landlord. No need to pass any parameters. Determine if the contract is activated. If not activated, it redirects to the activation page
- **addTenant():** Self-evaluated gas. Accessible by any users. Need to pass parameters of name, email, phone, nationality, and passport. Register the information into the contract. Once the landlord's information is registered, the contract code logic will default to the contract being activated.

### 3.2.8 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

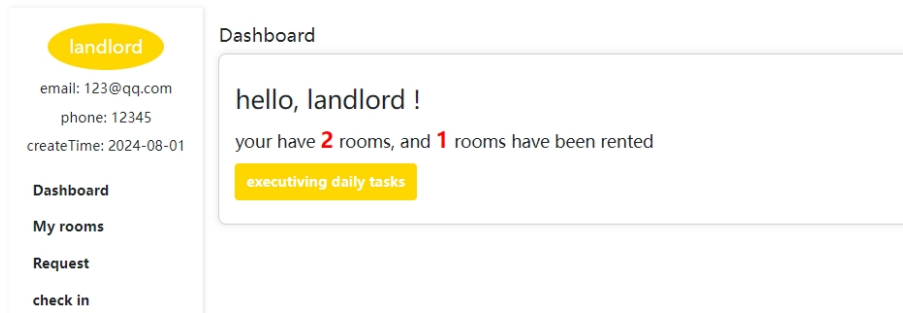
### 3.3.1 Applicable Personnel

- 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 dashboard. 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(the 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 interface:

- Panel 1 (Left):** Shows a room card for '400 x 200' with a red 'Not available!' stamp. The card details include: Address: Aberdeen, Causeway View; Block: 501D; Room type: Single; Price: £125.00 per week. It has 'close' and 'open' buttons.
- Panel 2 (Middle):** Shows a similar room card for '400 x 200' with details: Address: Aberdeen, Causeway View; Block: 501C; Room type: Single; Price: £125.00 per week. It also has 'close' and 'open' buttons.
- Panel 3 (Right):** Shows a form to add a new room. It includes input fields for: Address \*, Block \*, Room type \*, description \*, and Price \*: £. There is an 'addRoom' button at the bottom.

**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, users need to try again based on the prompt.

### 3.4.5 Main Flow

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

- **step 3:** If user 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 having 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

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:** Entner the address that the tenanct provide.
- **step 3:** Click the Searc 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

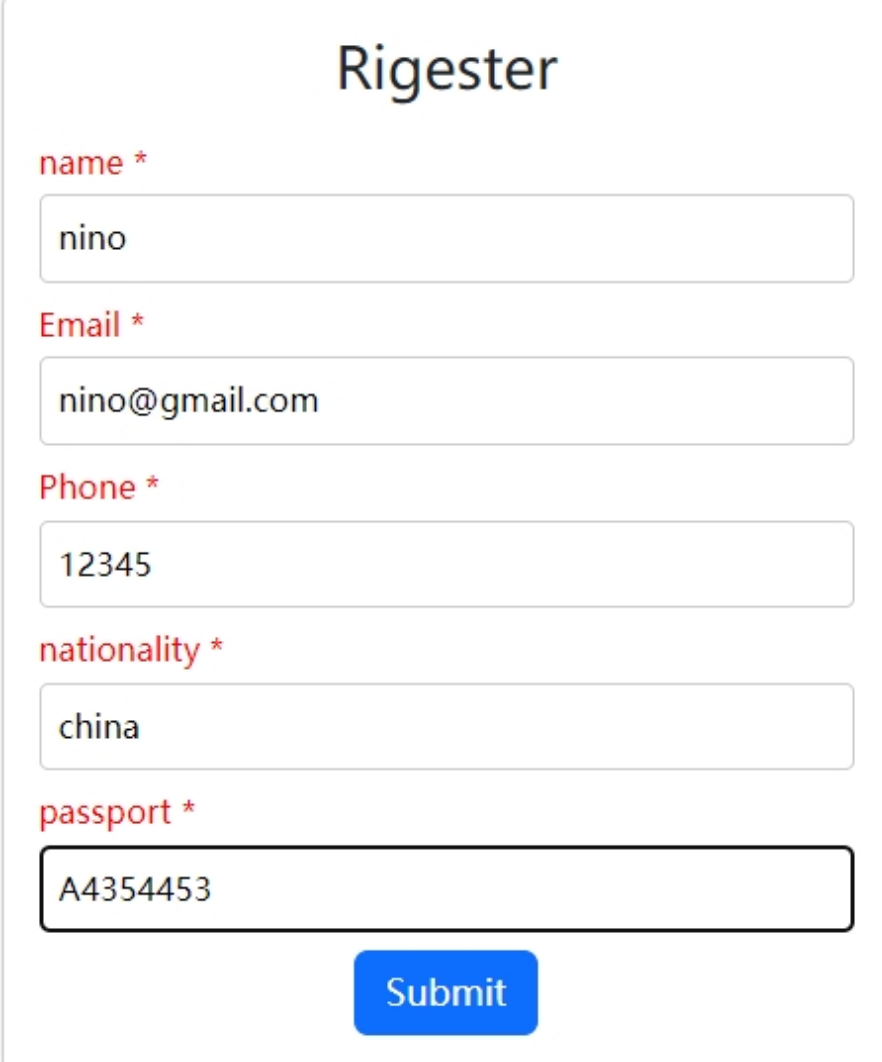
### 3.7.1 Applicable Personnel

- 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 the tenant 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

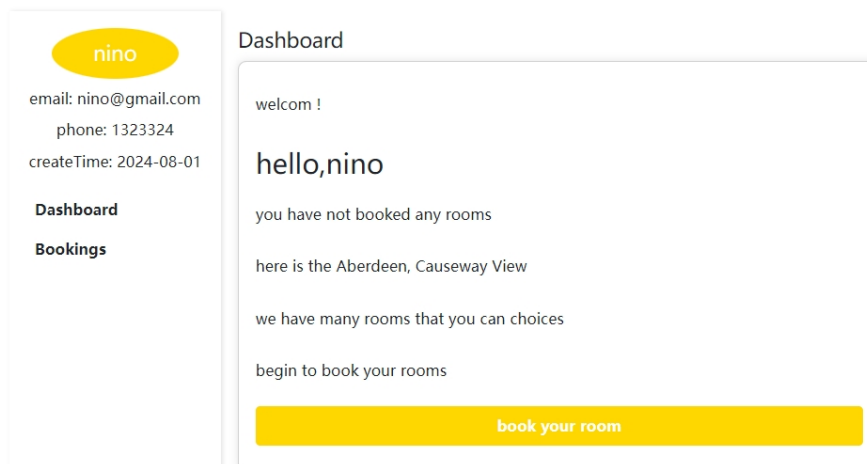
### 3.8.1 Applicable Personnel

- Tenant

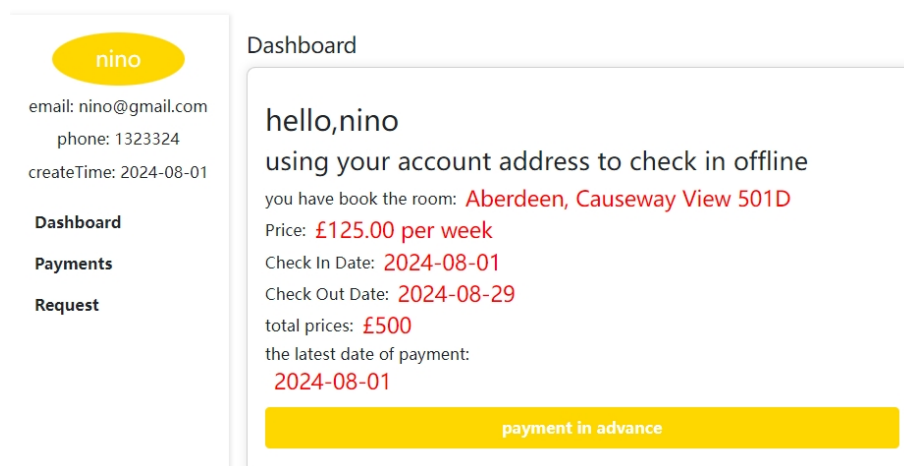
### 3.8.2 Preconditions

- 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 the tenant have not rented any room, tenant can click the book your room button to the room page. If tenant have already rented one room, tenant will get the detailed information of the tenancy, if tenant have pay the all cost, will show that tenant have pay all cost, but if tenant 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 tenant have already rented one room, tenant can check the detailed tenancy information.
- **step 3:** If tenant have already rented one room, tenant can click the payment in advance button to the payment page.
- **step 4:** if tenant 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(the 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(the Ethereum account address) and curTime. Determine whether tenant have rented one room by the account address and current time, if tenant 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

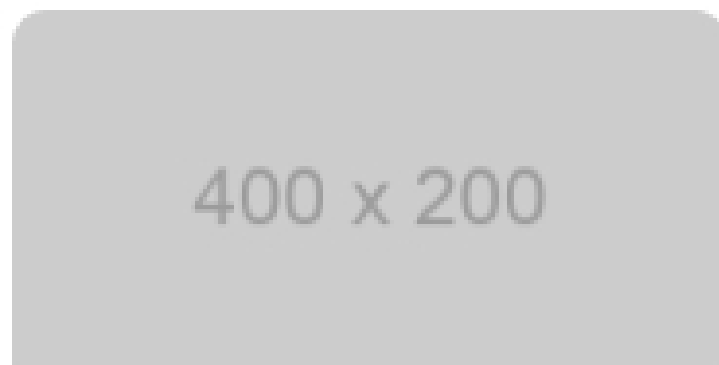
### 3.9.1 Applicable Personnel

- 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

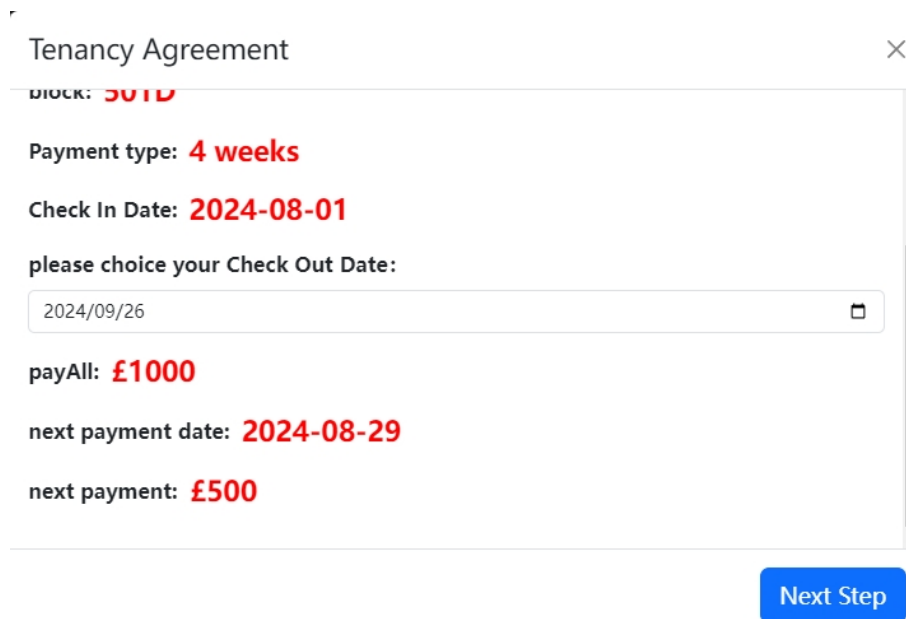
**Block:**501D

**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

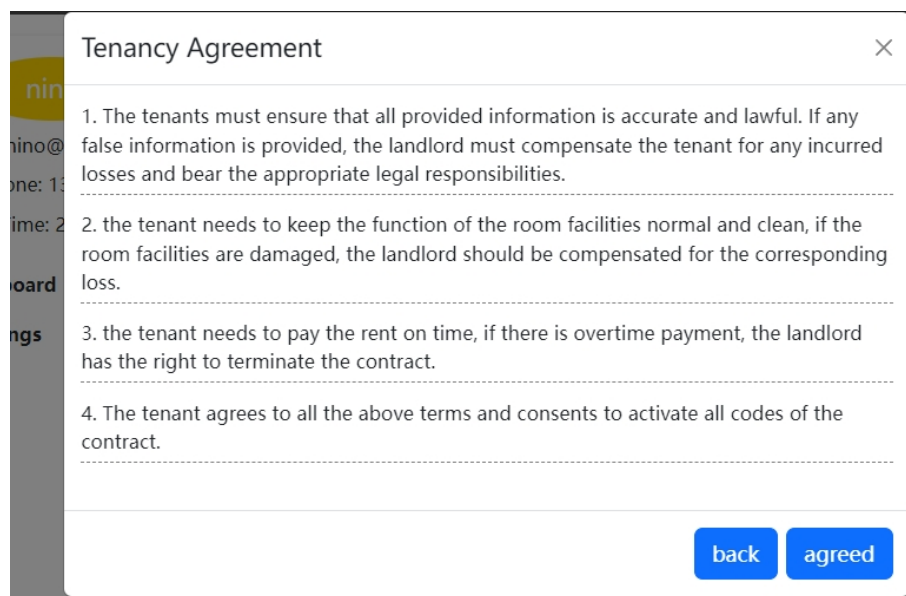
payAll: £1000

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

### 3.9.5 Main Flow

- **step 1:** Choice one room tenant want to rent, and click the book the room button.
- **step 2:** choice the 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

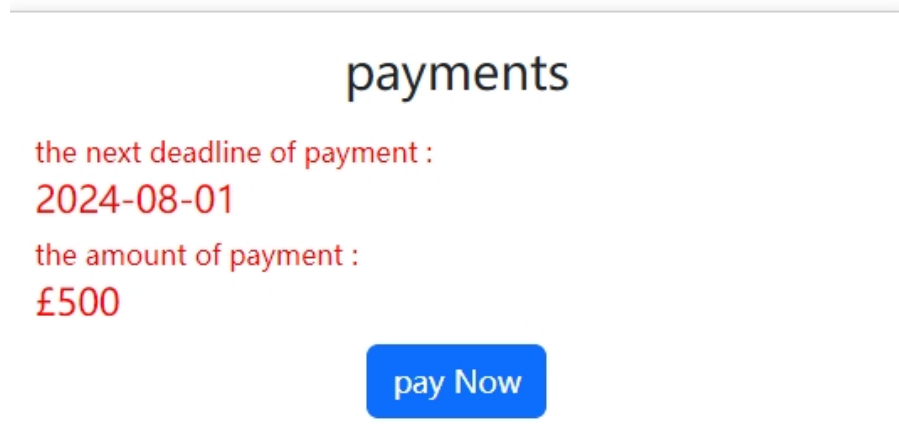


Figure 3.13: payment

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

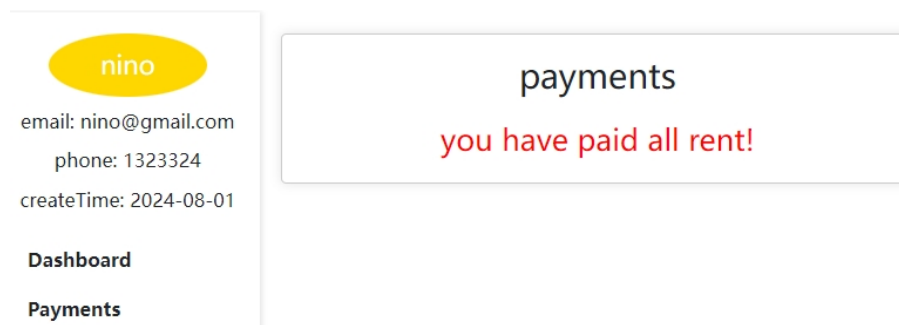


Figure 3.14: paid all payment

- - If after tenant 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 tenant 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 tenant 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

The screenshot shows a web interface for submitting a request. On the left, a user profile for 'nino' is displayed with contact details and a sidebar menu. The main form area on the right includes a category selector, a description text area, and a submit button.

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 tenant click the submit successfully, an successful pop-up box will be display, the page will remain, tenant can submit again.
- - after tenant 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 and Test guidance

## 4.1 System Deployment

### 4.1.1 Deployment Environment Configuration

- **Ganache:** Install Ganache, download Ganache from the Truffle Suite website (<https://trufflesuite.com/ganache/>), Download and install the version for the corresponding 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 the browser and install it.
- **Node.js (npm):** npm is integrated in node.js, so access the official Node.js website. Select the Node.js version for the corresponding 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, landlord 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 landlord can use the first address as the 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 provi
    lordAgreementTerms.push("3. The landlord must resolve any reasonabl
    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, modify the contract terms in the getAgreementTermsforlord and getAgreementTermsforuser methods according to the landlord's needs.

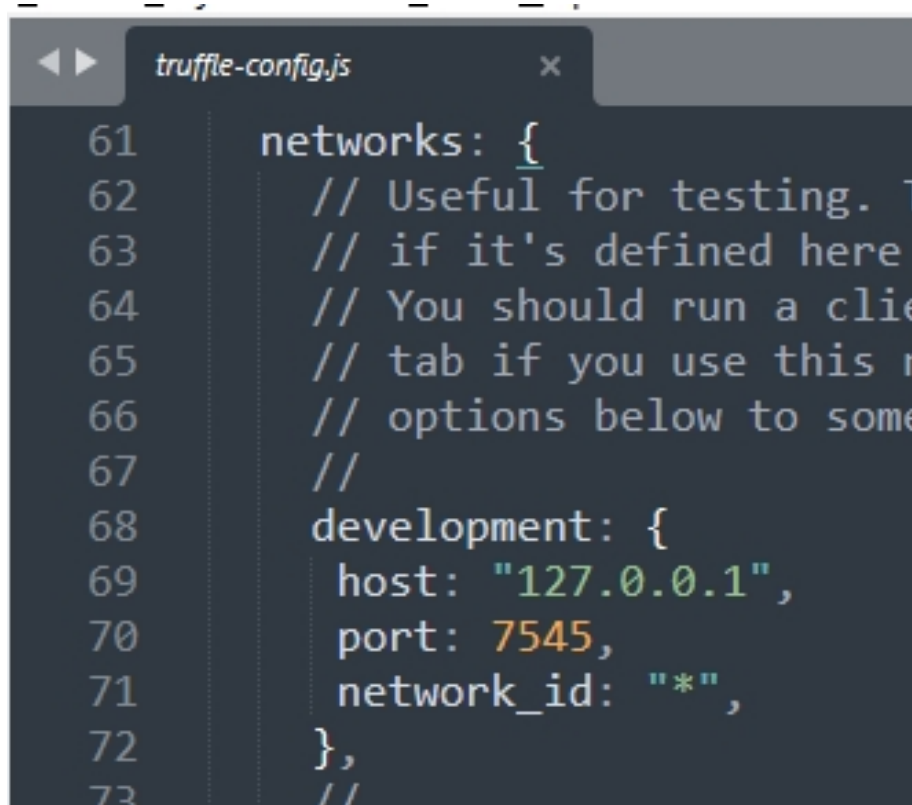
```

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, changing the address here to the landlord's address, which will be used for deployment.



```
61 networks: {
62   // Useful for testing.
63   // if it's defined here
64   // You should run a client
65   // tab if you use this network
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 the project to the local ganache network (based on the 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`. the 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.

## 4.2 System Test Guidance

Before testing, make sure the environment is configured and the contract is deployed.

### 4.2.1 Detailed steps

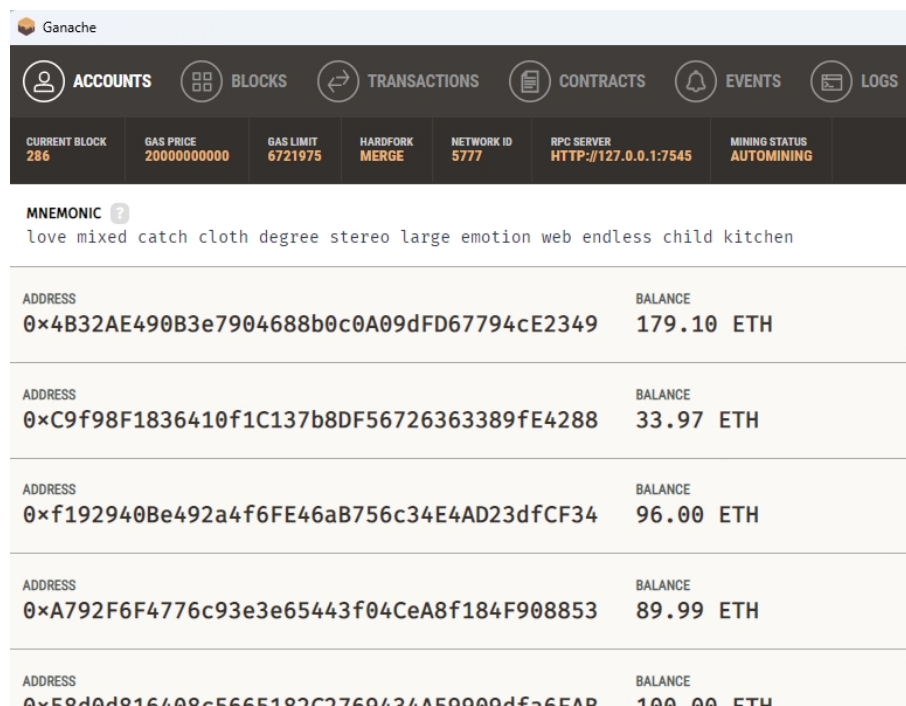


Figure 4.5: ganache

- **step1:** Startup the ganache, make sure the ganache local block network on.

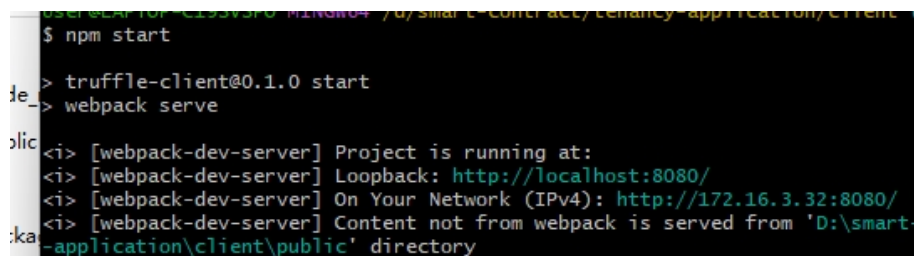


Figure 4.6: npm start

- **step2:** In the tenancy-application/client file directory, run command 'npm start'. can get the webpage link in the command box.
- **step3:** Open three browsers. Use the first browser to connect account A (landlord), the second browser to connect account B (tenant), and the third browser to connect account C (tenant).
- **step4:** Begin to test the functionality based on the chapter3 of System Functional Module Description.



## Chapter 5

# System Security Analysis

### 5.1 Information Reliability Analysis

Information reliability is crucial in maintaining the integrity and trustworthiness of the Blockchain-based Smart Contract System for Tenancy. This section evaluates the mechanisms and protocols to analyse how the information processed and stored within the system is accurate and reliable.

#### 5.1.1 Data Validation and Verification

```
function addTenant(address tenantAddress, uint256 createTime, string memory
    require(tenantAddress != address(0), "Tenant address cannot be empty");
    require(createTime > 0, "Create time must be a valid number");
    require(bytes(name).length > 0, "Name cannot be empty");
    require(bytes(email).length > 0, "Email cannot be empty");
    require(bytes(phone).length > 0, "Phone cannot be empty");
    require(bytes(nationality).length > 0, "Nationality cannot be empty");
    require(bytes(passport).length > 0, "Passport cannot be empty");

    // Simplified email and phone format checks
    require(isValidEmail(email), "Invalid email format");
    require(isValidPhone(phone), "Invalid phone format");
    uint256 tenantId = tenants.length + 1;
    tenants.push(Tenant({
        tenantId: tenantId
```

Figure 5.1: the sanitization of smart contract

```
const validateEmail = (email) => {
    const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return re.test(String(email).toLowerCase());
};

const validatePhone = (phone) => {
    const re = /^+?[1-9]\d{1,14}$/; // E.164 国际号码格式
    return re.test(String(phone));
};

const validateForm = () => {
    const { email, name, phone, nationality, passport } = formValues

    if (!validateEmail(email)) return false;
    if (!validatePhone(phone)) return false;

    //
    if (!name || !nationality || !passport) return false;


    return true;
};
```

Figure 5.2: the validation of front-end


- **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. For the front end, each input field needs to be validated, and only when all the validation is passed can the next operation be carried out. For smart contract code, each input data also needs to be verified by require method.

http://localhost:8080

0xa48bb...04541 CONTRACT INTERACTION ⓘ



DETAILS HEX

Estimated fee  0.00052846  
**0.00052846 ETH**  
Site suggested -15 sec Max fee: 0.00052846 ETH

Total 0.00052846  
**0.00052846 ETH**  
Amount + gas fee Max amount: 0.00052846 ETH

Reject Confirm

**Figure 5.3:** the validation of front-end

- **Transaction Verification:** Each transaction that could modify the information 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 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. Such as room information and contract terms, once they are written to the blockchain, they cannot be modified, and all users get the same data information.

## 5.2 Information Security Analysis

Information security is paramount in protecting the smart 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

```
const handleEncrypt = () => {  
  const secretKey = formValues.name+accounts[0];  
  const ciphertext = CryptoJS.AES.encrypt(formValues.passport, secretKey).toString();  
  setEncryptedText(ciphertext);  
};  
  
const handleDecrypt = () => {  
  const secretKey = formValues.name+accounts[0];  
  const bytes = CryptoJS.AES.decrypt(formValues.passport, secretKey);  
  const originalText = bytes.toString(CryptoJS.enc.Utf8);  
  setDecryptedText(originalText);  
};
```

**Figure 5.4:** the code of encrypt

- **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. In this system, the front end will use the user's name plus his account address as his decryption key, for the front end, when the customer stores important information such as passport number, the front end will use encryption algorithm technology to encrypt this data.
- **Secure Communication:** All communications between users and landlord are secured using Transport Layer Security (TLS) protocols to prevent eavesdropping and man-in-the-middle attacks. In addition, the user's personal privacy is completely confidential during all transactions.

### 5.2.2 Access Control

```

modifier onlyLandlord() {
    require(msg.sender == landlordAddress, "Only landlord can perform this");
}

modifier oncePerDay() {
    require(block.timestamp >= lastExecutionTime + 1 days, "Action can only be performed once per day");
}

modifier onlyOnce() {
    require(!isInitialized, "This function can only be called once");
}

function dailyAction(uint256 curTime) public onlyOnce | onlyLandlord {
    lastExecutionTime = curTime;

    for (uint i = 0; i < rentInfos.length; i++) {
        if (rentInfos[i].endTime < lastExecutionTime) {
            updateRentInfo(rentInfos[i].rentId); // Expired, update this
        }
    }
}

```

Figure 5.5: the access authorization

- **Authentication:** Multi-factor authentication (MFA) is required for accessing the system, providing an additional layer of security beyond passwords. In this system, in addition to verifying whether the user is connected to MetaMask, it is also necessary to verify whether the user has registered his or her information and whether the user is landlord or tenant. Verify the user's information status, etc. In addition, although some validations are derived from the smart contract code, they can be guaranteed to be absolutely objective and fair.
- **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. In this system, the roles are mainly landlord and tenant. In the smart contract code, different methods limit the access of different roles. In addition, some methods limit the access of different roles as well as the number of visits per day.

### 5.2.3 Event Response and recording

```

event RentPaid(address indexed tenant, uint256 roomId, uint256 amount, uint256 timestamp);
event RoomStatusUpdated(uint256 roomId, string newStatus, uint256 timestamp);
event MaintenanceRequested(uint256 requestId, uint256 roomId, string requestType, string description);
event MaintenanceResolved(uint256 requestId, uint256 timestamp);

```

Figure 5.6: the event code

- **Recording:** record of system activity through security information and event management (SIEM) systems helps in detecting anomalies and potential security incidents in real time. In this system, a number of event methods are also written in the smart contract code, which are used to record and track specific activities or state changes, and these events also help users track important activities and changes in the execution of contracts on the blockchain.

```
// Monitor the RentPaid event
contract.events.RentPaid({})
  .on('data', (event) => {
    console.log('RentPaid Event:', event);
    // if successful
    alert('payment successful!')
  })
  .on('error', (error) => {
    // console.error('Event Error:', error);
  });
```

Figure 5.7: the front-end response

- **Event Response:** A well-defined Event response plan is in place, enabling quick identification, containment, and remediation of security breaches. In this system, the front end will monitor the events of the contract if any accidents occur on the way to the contract interaction. The front end can immediately monitor the accident and make corresponding processing

## 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 Blockchain-based Smart Contract System for Tenancy 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

```
string[] private lordAgreementTerms;  
function getAgreementTermsforlord() public onlyLandlord returns (string[] memory) {  
    lordAgreementTerms.push("1. The tenants must ensure that all provided information  
    lordAgreementTerms.push("2. the tenant needs to keep the function of the room fac  
    lordAgreementTerms.push("3. the tenant needs to pay the rent on time, if there is  
    lordAgreementTerms.push("4. The tenant agrees to all the above terms and consents  
    return lordAgreementTerms;  
}  
  
string[] private userAgreementTerms;  
function getAgreementTermsforuser() public onlyLandlord returns (string[] memory) {  
    userAgreementTerms.push("1. The landlord must update daily tasks every day. If the  
    userAgreementTerms.push("2. The landlord must ensure that all provided information  
    userAgreementTerms.push("3. The landlord must resolve any reasonable requests subm  
    userAgreementTerms.push("4. If the landlord delays or neglects to address tenant r  
    userAgreementTerms.push("5. The landlord agrees to all the above terms and consent  
    return userAgreementTerms;  
}
```

Figure 5.8: the front-end response

- **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. In this system, the contract agreement between the landlord and the tenant is directly written in the smart contract code, and this information is not deployed once. It's never gonna change
- **Auditability:** The immutability of smart contracts allows for complete audit trails. Tenants and landlords can view the audit contract code at any time, ensuring the transparency of information.

## 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. In the process of developing this system, I did encounter many such problems. Because the testing process of smart contract code is not perfect and inefficient. So when developer deploy this contract, developer will maybe find some bugs that developer didn't find during the testing, and also developer may find many functionality that lack certain attributes, or that the logic of certain methods is actually not perfect. At this point, if developer re-deploy after reworking the code, developer need to start testing the functionality again. Because the contract for the new deployment is no longer related to the original one.
- **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 system. during the process of testing this system, I found that when I used multiple users at the same time, or when I frequently operated a function, I had to wait a long time to get a notification that the transaction had been completed, which really greatly reduced user satisfaction.
- **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. Since the smart contract code is available for all users to see, it cannot be modified once deployed. So if developers enter some personal information in a piece of code, or because of defects in the code, some methods may expose users' privacy. Due to the immutability of blockchain, which could lead to an unsolvable breach of user privacy.

### 6.1.2 Operational Challenges

- **User Adoption:** Despite the clear advantages of the Blockchain-based Smart Contract System for Tenancy, there may be resistance from users accustomed to traditional tenancy markets. Educating and encouraging users to understand and embrace this new technology requires significant effort and time. After the development of this system, I asked a few of my friends to use this system, but most of them think this system is too cumbersome, because they do not understand the underlying mechanism of this system. They don't feel the security of the system. They all be more willing to use traditional tenancy methods.
- **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. Many developing countries do not have laws specifically for smart contracts.

## 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 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.



## Chapter 7

# Conclusions

In conclusion, the implementation and adoption of Blockchain-based Smart Contract System for Tenancy represent a significant advancement in the tenancy 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.

Blockchain-based Smart Contract System for Tenancy 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

- [1] B S Anupama and N R Sunitha. Analysis of the consensus protocols used in blockchain networks – an overview. In *2022 IEEE International Conference on Data Science and Information System (ICDSIS)*, pages 1–6, 2022.
- [2] The Solidity Authors. Solidity. <https://docs.soliditylang.org/en/v0.8.26/>, 2023. Accessed: 2024-08-08.
- [3] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.
- [4] Qiang Deng and Renfei Luo. Blockchain consensus algorithm for intelligent cultural tourism based on trust mechanism. In *2023 3rd International Conference on Computer Science and Blockchain (CCSB)*, pages 182–185, 2023.
- [5] Ethereum Revision f3846d1c. web3js. <https://web3js.readthedocs.io/en/v1.10.0/>, 2016. Accessed: 2024-08-08.
- [6] A Consensys Formation. Metamask. <https://learn.metamask.io/overview>, 2024. Accessed: 2024-08-08.
- [7] ConsenSys Software Inc. Truffle. <https://archive.trufflesuite.com/docs/truffle/>, 2016. Accessed: 2024-08-08.
- [8] ConsenSys Software Inc. Ganache. <https://archive.trufflesuite.com/docs/ganache/>, 2022. Accessed: 2024-08-08.
- [9] Anushka Jain, Aparna Gangwar, Chandni Kumari, and Pawan Singh Mehra. A survey on blockchain for rental lease management. In *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 1–5, 2024.
- [10] Ran Ma, Xiaotian Yang, and Fei Gao. Discussion on smart contract under blockchains technology. In *2022 International Conference on Industrial IoT, Big Data and Supply Chain (IIoTBDSC)*, pages 338–342, 2022.
- [11] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access*, 7:117134–117151, 2019.
- [12] Jeidy Panduro-Ramirez, Ashvine Kumar Sharma, Gurpreet Singh, Kumari H. Pavana, Claudia Poma-Garcia, and Surendra Kumar Shukla. Blockchain implementation in financial sector and cyber security system. In *2022 11th International Conference on System Modeling Advancement in Research Trends (SMART)*, pages 754–760, 2022.
- [13] Chitturi Prasad, Gummuluri Udaya Chandrika, Vani Venkata Sai Sindhu Garapati, Ganaga Rama Koteswara Rao, Vadavalli Harshitha Manaswini, and Pinnamaneni Meghana.

- Quantifying blockchain immutability over time. In *2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 715–720, 2021.
- [14] Kavita Saini, Abhishek Roy, Pethuru Raj Chelliah, and Tanisha Patel. Blockchain 2.0: A smart contract. In *2021 International Conference on Computational Performance Evaluation (ComPE)*, pages 524–528, 2021.
- [15] @sawaratsuki1004. React. <https://react.dev/learn>, 2024. Accessed: 2024-08-08.