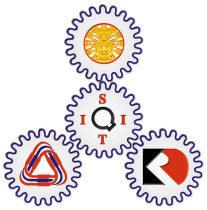


SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION TECHNOLOGY(ICT), SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY, THAMMASAT UNIVERSITY

CAPSTONES PROJECT REPORT

CSS486 - BLOCKCHAIN DEVELOPMENT

GROUP PROJECT: RENTING DECENTRALIZED APPLICATION

SUBMITTED TO

Dr.Watthanasak Jeamwatthanachai

BY

Natthaphol Pornpholkullapat ID.6222781898

Warit Phankawee ID.6222782383

Tamasit Savanpopan ID.6222781815

TABLE OF CONTENTS

Chapter 1

Introduction

Main Problem

A significant issue in renting a house or room is dealing with dishonest or non-paying renters. This problem arises when a tenant fails to fulfill their financial obligations, such as not paying rent or causing damage to the property. To address this problem, implementing a well-defined rental agreement or contract can provide protection for both landlords and tenants. The contract can include clauses related to payment terms, security deposits, lease duration, maintenance responsibilities, and penalties for non-payment or early termination.

By clearly outlining the terms and conditions in the rental contract, it becomes easier to hold renters accountable for their financial obligations. It provides a legal framework for resolving disputes, recovering unpaid rent, and taking appropriate action against non-compliant tenants.

In the context of decentralized applications (DApps) and blockchain technology, a smart contract can be utilized to automate and enforce the terms of the rental agreement. By leveraging the transparency, immutability, and self-executing nature of smart contracts, it becomes more difficult for renters to default on their payments or breach the terms of the agreement without consequence.

For example, a smart contract could hold the tenant's funds in escrow and release them to the landlord automatically on the specified due dates. If the tenant fails to make the payment, the smart contract can trigger penalties or initiate a resolution process.

Integrating such a smart contract solution into the rental process can mitigate the risk of non-payment and provide a more secure and efficient method for landlords and tenants to transact.

Objectives

The primary objective of this blockchain-based renting platform aims to streamline the rental process, establish trust between landlords and renters, and enhance financial security. Through the utilization of smart contracts and decentralized technology, the platform brings forth advantages like transparency, automated rent transactions, and increased accountability in the rental market. Its main goal is to create a more efficient and reliable rental ecosystem where all parties involved can benefit from a seamless and secure renting experience.

Chapter 2

Related Documents and Literature

A Blockchain-based Housing Rental System

The paper proposes a blockchain-based solution for a housing rental system that aims to enable peer-to-peer sharing of listings information without the need for intermediaries. The solution utilizes blockchain technology, specifically Ethereum and IPFS (Inter Planetary File System), along with smart contracts to improve the efficiency of the leasing process and provide legal protection for tenants and landlords.

The main contributions of the proposed system are as follows:

Use of blockchain technology: The system leverages blockchain technology to ensure secure and tamper-proof sharing of listings information without the need for a central authority. By storing the hash of the listings information on the blockchain, the integrity and authenticity of the data are maintained.

Smart lease contracts: Traditional lease agreements are programmatically converted into smart lease contracts using Ethereum's Smart Contract functionality. This automation streamlines the leasing process and provides a transparent and traceable record of transactions.

Integration of IPFS: IPFS is utilized as a decentralized storage system for the listings information. The actual content of the listings, such as images, is stored on IPFS, while the hash of the content is stored on the blockchain. This approach ensures efficient storage and retrieval of data while maintaining data integrity.

The core operating mechanism of the system involves interactions between Ethereum Smart Contracts, IPFS, and participants. The landlord uploads the listings information in JSON format to IPFS, which returns a hash representing the information. The hash is then stored in the listings contract on the blockchain. Tenants can access the hash from the listings contract and retrieve the listings from IPFS. If the landlord and tenant agree, a smart lease

contract is created through the lease agreement producer contract, and its address is stored for future reference.

The smart contracts in the system are categorized into three types: listings contract, lease agreement producer contract, and smart lease contract. The listings contract stores the hashes of the listings information and controls the availability of the information. The lease agreement producer contract handles the creation of smart lease contracts based on lease process information provided by the landlord. The smart lease contract represents the lease agreement and manages the various statuses and transactions throughout the lease period, including rental payments, deposit refunds, and generation of rental bills.

The proposed system aims to address the limitations of traditional housing rental systems, such as centralized control, lack of transparency, and potential coordination disputes. By leveraging blockchain technology and smart contracts, it offers a decentralized and transparent solution that enhances the efficiency and security of the housing rental process**.**

Ganache

- Ganache is a versatile and powerful development tool specifically designed for Ethereum blockchain developers. It provides a local blockchain environment that allows developers to deploy, test, and debug smart contracts with ease. By simulating the Ethereum network locally, Ganache eliminates the need for developers to interact with the live blockchain during the development and testing phases, providing a more efficient and controlled environment.

Key details and features of Ganache:

- **Local Blockchain Simulation:** Ganache creates a local Ethereum network that mimics the behavior of the live Ethereum blockchain. This simulated network allows developers to interact with it using their own custom configurations and test scenarios without incurring any costs or affecting the main Ethereum network.

- **Customizable Network Parameters:** Ganache offers extensive customization options, allowing developers to configure various network parameters. This includes gas limits, block time, account balances, network ID, and chain ID. Developers can simulate different network conditions and test their smart contracts under various scenarios.
- **Account Management:** Ganache provides a set of pre-funded accounts for testing purposes. These accounts come with preloaded Ether (ETH), allowing developers to deploy and interact with their smart contracts seamlessly. Developers can also import external accounts or create new ones for testing specific use cases.
- **Transaction Control and Inspection:** Ganache allows developers to have fine-grained control over transactions. Developers can set the gas price, gas limit, and nonce for each transaction. They can also inspect the status, gas usage, events emitted, and other transaction details for debugging and analysis purposes.
- **Debugging and Logging:** Ganache offers robust debugging capabilities for smart contracts. It provides detailed transaction information, including stack traces and events emitted during contract execution. Developers can leverage these features to identify and resolve issues in their smart contracts more efficiently.
- **Integration and Tooling:** Ganache seamlessly integrates with popular Ethereum development frameworks such as Truffle and Remix. This integration streamlines the development workflow and provides additional features like contract compilation, deployment, and testing automation. Developers can also integrate Ganache with Ethereum wallets and clients to interact with the local blockchain easily.
- **Testing and Security Analysis:** Ganache facilitates automated testing of smart contracts. It supports testing frameworks like Truffle and provides functionalities to create comprehensive test suites, enabling developers to verify the correctness and robustness of their contracts before deployment.

Flask

- Flask is a popular web framework for Python that provides a simple and flexible way to build web applications. It is known for its minimalistic design, ease of use, and extensive ecosystem. Flask follows the microframework approach, focusing on simplicity and giving developers the freedom to choose their tools and libraries. Here are the key details and features of Flask:
- **Lightweight and Minimalistic:** Flask is designed to be lightweight and minimalistic, providing just the essentials for web development. It has a small core and relies on external libraries to add functionality as needed. This minimalist approach allows developers to have more control over their application's structure and dependencies.
- **Routing and URL Mapping:** Flask uses a routing mechanism that maps URLs to Python functions, known as view functions. Developers can define routes using decorators or explicit routing rules, making it easy to handle different URL patterns and HTTP methods. This flexibility enables the creation of clean and organized URL structures.
- **Template Engine:** Flask comes with a built-in template engine called Jinja2, which allows developers to create dynamic HTML pages. Jinja2 provides features like template inheritance, macros, and filters, making it easy to generate HTML content dynamically based on data from the application.
- **Request-Response Cycle:** Flask handles the request-response cycle efficiently. When a user makes a request, Flask processes the request, executes the appropriate view function, and generates a response that is sent back to the user's browser. Flask supports handling different types of requests, including GET, POST, and more.
- **Flask Extensions:** Flask has a rich ecosystem of extensions that enhance its functionality. These extensions cover various areas such as database integration, form validation, user authentication, API development, and more. Developers can choose and integrate these extensions based on their specific requirements, reducing development time and effort.
- **Built-in Development Server:** Flask provides a built-in development server, making it easy to test and run applications during the development phase. The development server automatically reloads the application when changes are detected, allowing developers to see the updates in real-time. However, for production deployments, it is recommended to use a more robust web server.

- **Flask-WTF (Forms):** Flask-WTF is a Flask extension that simplifies form handling and validation. It integrates with Flask and provides a convenient way to define forms in Python classes. Developers can use Flask-WTF to generate HTML forms, handle form submissions, and perform validation on user input.
- **Flask-SQLAlchemy (Database Integration):** Flask-SQLAlchemy is a Flask extension that simplifies database integration using SQLAlchemy, a popular Python SQL toolkit and Object-Relational Mapping (ORM) library. It provides an easy-to-use interface for interacting with databases, allowing developers to define models, execute queries, and manage database operations efficiently.
- **Flask-RESTful (API Development):** Flask-RESTful is an extension that simplifies the creation of RESTful APIs with Flask. It provides tools and decorators to define API resources, handle request parsing, and serialize/deserialize data in various formats like JSON. Flask-RESTful streamlines the development of robust and scalable API endpoints.
- **Flask-Login (User Authentication):** Flask-Login is a Flask extension that simplifies user authentication and session management. It provides a flexible and secure way to handle user logins, logouts, and session persistence. Flask-Login integrates seamlessly with Flask applications, enabling developers to protect routes and implement user-specific functionalities.

Web3

- Web3, also known as Web3.0 or Web 3, refers to the next generation of the internet that aims to revolutionize how we interact with digital services and assets. It is a set of protocols and technologies that leverage blockchain and decentralized technologies to create a more open, secure, and user-centric web. Here are the key details and concepts related to Web3:
- **Decentralization:** Web3 is built upon the principle of decentralization, which means that power and control are distributed across a network of participants rather than being concentrated in a central authority. Blockchain technology plays a crucial role in enabling decentralization by providing a transparent and tamper-proof ledger where data and transactions are recorded.

- **Blockchain Technology:** At the core of Web3 lies blockchain technology, which is a distributed and immutable ledger that records transactions and data across a network of computers. Blockchain ensures transparency, security, and trust by eliminating the need for intermediaries and enabling peer-to-peer interactions.
- **Smart Contracts:** Web3 utilizes smart contracts, which are self-executing contracts with predefined rules and conditions. These contracts are stored and executed on a blockchain network, enabling automated and trustless interactions. Smart contracts enable a wide range of applications, including decentralized finance (DeFi), non-fungible tokens (NFTs), and decentralized applications (dApps).
- **Cryptocurrencies and Tokens:** Web3 embraces cryptocurrencies and tokens as digital assets that can be exchanged, stored, and transferred securely without the need for traditional intermediaries. Cryptocurrencies like Bitcoin and Ethereum play a significant role in powering Web3 networks, enabling seamless value transfer and incentivizing network participants.
- **Interoperability and Standards:** Web3 promotes interoperability between different blockchain networks and protocols, allowing seamless communication and data exchange. Standards such as the ERC-20 and ERC-721 for Ethereum-based tokens, as well as protocols like IPFS (InterPlanetary File System) for decentralized storage, facilitate interoperability and enhance the overall user experience.
- **Web3 Browsers and Wallets:** Web3 browsers and wallets are specialized software applications that enable users to interact with Web3 networks and decentralized applications. These tools provide access to blockchain networks, facilitate transactions, manage digital assets, and offer enhanced security measures like private key management.
- **User Empowerment and Data Ownership:** Web3 aims to empower individuals by giving them control over their data and digital identities. Users can securely store their personal information and selectively grant access to applications and services, enhancing privacy and reducing the reliance on centralized data repositories.
- **Web3 Applications (dApps):** Web3 promotes the development of decentralized applications (dApps), which are applications that run on a distributed network rather than a centralized server. dApps leverage blockchain technology, smart contracts, and Web3 protocols to provide transparent, secure, and censorship-resistant solutions across various industries.

- **Governance and Consensus:** Web3 networks often employ decentralized governance models, allowing network participants to have a say in the decision-making process. Consensus mechanisms like Proof of Stake (PoS) and Delegated Proof of Stake (DPoS) ensure the security and validity of transactions by involving network participants in block validation and network governance.
- **Future Potential:** Web3 has the potential to transform numerous industries, including finance, supply chain, healthcare, gaming, and more. It can enable new business models, enhance security and privacy, foster financial inclusion, and redefine the relationship between users and digital services.

Chapter 3

Design and Implementation

Requirements

User Registration:

- The platform should allow landlords and renters to sign up by providing the required information.

Dormitory Management:

- The platform should offer landlords the capability to add dormitories, providing essential information such as the dormitory name, location, and the number of available rooms.
- Every dormitory will keep a record of its rooms, encompassing details such as the prices and deposit amounts associated with each room.

Room Availability and Booking:

- Renters should have the ability to explore and view the list of available rooms in various dormitories on the platform.
- Renters should be provided with the option to choose a room and start the booking process, taking into consideration the availability of the selected room.

Rent Payments and Deposits:

- Renters should be able to deposit the required amount for rent and security deposit through the platform using a secure payment method.
- The platform should securely hold the rent and deposit amounts until the rental agreement is terminated.

Rent Withdrawals for Landlords:

- The platform should enable renters to securely submit the necessary payment for rent and security deposit using a trusted payment method.

- The withdrawal process should prioritize security and transparency, ensuring that transactions are conducted in a secure manner and maintaining a transparent record of all withdrawals.

Rental Agreement Management:

- The platform should streamline the process of creating and overseeing rental agreements between landlords and renters, offering efficient management and administration of these agreements.
- Rental agreements should encompass specific information, including the duration of the rental, terms and conditions, as well as payment schedules.

Termination and Dispute Resolution:

- The platform should effectively manage the termination of rental agreements, allowing both landlords and renters to initiate the termination process based on predetermined rules. In the event of disputes or problems, the platform should incorporate a conflict resolution mechanism to ensure equitable resolutions.

Security and Privacy:

- The platform should employ strong security measures to safeguard user data, financial transactions, and any sensitive information, ensuring the utmost protection and confidentiality.
- The platform should prioritize user privacy by securely storing personal information and ensuring that it is accessible solely to authorized individuals or entities.

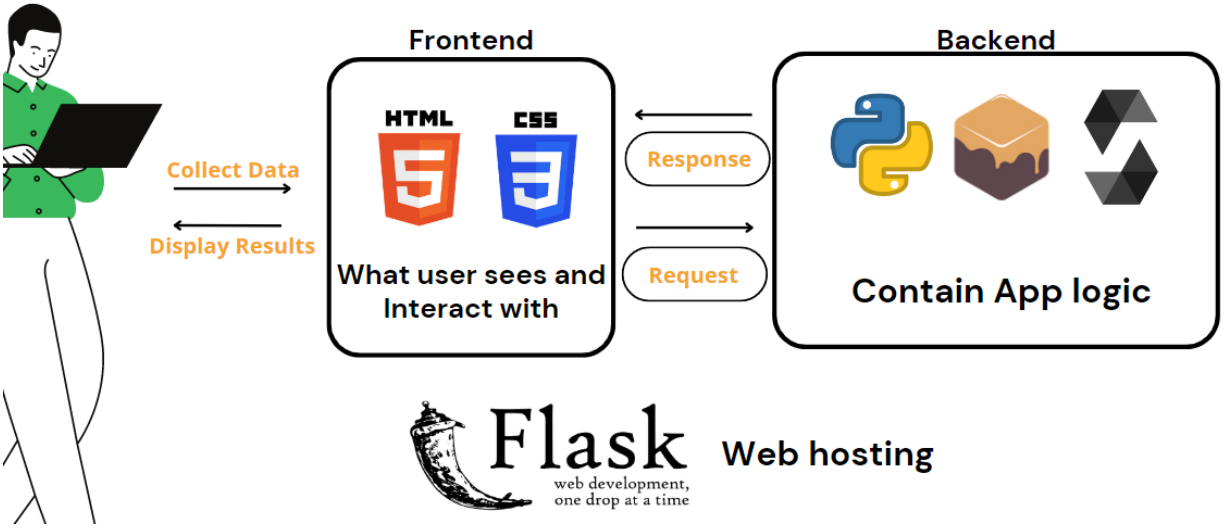
User Interface and Experience:

- The platform should offer a user-friendly and intuitive interface, designed to enhance the interaction between landlords and renters. It should provide a seamless and efficient user experience, enabling easy navigation, room selection, and effective management of transactions.

Testing and Quality Assurance:

- The platform should undergo comprehensive testing to detect and address any potential bugs, vulnerabilities, or usability concerns. Regular quality assurance evaluations should be conducted to ensure the platform's dependability and stability.

System Architecture



The architecture consists of three main components: the frontend, the backend, and the smart contracts.

- Frontend: HTML/CSS. User interface of the web application. HTML structures the content, and CSS styles the presentation. Users interact with the frontend by inputting data and interacting with various elements.
- Backend: Python, Ganache, and Flask Powers the web application's brain. Python handles core functionalities. Ganache acts as a local Ethereum blockchain network for development and testing. Flask serves as the web server and facilitates communication between the frontend and the blockchain network.
- Smart Contracts: Solidity. Written in Solidity, a programming language for Ethereum. Self-executing contracts with predefined rules and conditions. Define the business logic, data storage, and rules for the blockchain application.

User Interaction: Users input data and interact with the frontend interface. Frontend captures user input and initiates a request to the backend.

Backend Processing: Backend powered by Python and Flask, receives the user's data request. Processes the request, validates input, and prepares data for interaction with the blockchain network.

Blockchain Interaction: Backend interacts with the local blockchain network (Ganache) using deployed Solidity smart contracts. Executes predefined actions by invoking functions in the smart contracts.

Blockchain Transparency and Security: Transactions recorded on the blockchain ensure transparency, immutability, and decentralization. Consensus mechanism and cryptographic algorithms maintain data integrity.

Data Presentation and Feedback: Frontend updates interface reflecting the updated state of data after the transaction is confirmed on the blockchain. Provides feedback to the user, such as success messages or error notifications.

By combining HTML/CSS frontend, Python with Ganache and Flask in the backend, and Solidity smart contracts, we create a secure, transparent, and decentralized blockchain web application.

Smart Contract

Structs:

- Landlord: Stores information about a landlord, including their dormitory names, balance, and whether they are a registered landlord.
- Dorm: Represents a dormitory and maintains a list of rooms along with their details.
- Renter: Holds information about a renter, such as the address of the associated landlord, deposit amount, monthly payment, last payment time, active status, balance, warning status, and termination pending status.
- Room: Contains information about a room, including its ID, price, existence status, deposit amount, and availability.

Mappings:

- dorms: Maps the name of a dormitory to its corresponding Dorm struct.
- landlords: Maps the address of a landlord to their Landlord struct.
- renters: Maps the address of a renter to their Renter struct.
- dormToLandlord: Maps the name of a dormitory to the address of its associated landlord.
- dormRooms: Maps the name of a dormitory and room ID to the corresponding Room struct.

Arrays:

- dormNames: Stores the names of all registered dormitories.
- activeRenters: Contains the addresses of active renters.
- pendingTermination: Holds the addresses of renters who have requested contract termination.

Variables:

- rentDueInterval: Represents the duration between rent payments.

Modifiers:

- onlyRenter: Restricts the execution of a function to only active renters.
- onlyLandlord: Restricts the execution of a function to only registered landlords.

Functions:

- registerLandlord: Allows a landlord to register on the platform by providing their name. Creates corresponding entries in the landlords mapping and dormToLandlord mapping.
- registerRenter: Enables a renter to register by specifying the dormitory name and room ID they wish to rent. Verifies the deposit amount and adds the renter to the renters mapping.
- getAllDormNames: Retrieves an array of all registered dormitory names.
- getAvailableRooms: Returns an array of available room IDs for a given dormitory.
- getBalanceRenter/getBalanceLandlord: Retrieves the balance of a renter or landlord, respectively.
- addRoom: Allows a landlord to add a room to their dormitory by specifying the dormitory name, price, and deposit amount.
- getRoomPrice: Returns the price and deposit amount for a specific room in a dormitory.
- depositRent/withdrawRent/withdrawBalance: Handle the deposit and withdrawal of funds for renters and landlords.
- terminateContractRenter/terminateContractLandlord: Facilitates the termination of a rental agreement by either the renter or the landlord.
- getPendingTermination: Retrieves an array of renters whose contract termination is pending landlord confirmation.
- terminateContractAuto: Automatically terminates a rental contract when a renter fails to pay rent.
- autopay: Automatically deducts monthly rent from active renters' balances and pays it to the associated landlords.

Helper Functions:

- checkifinList: Checks if a given dormitory name exists in a list of dormitory names.
- removeElement: Removes a specific element from an array of addresses.

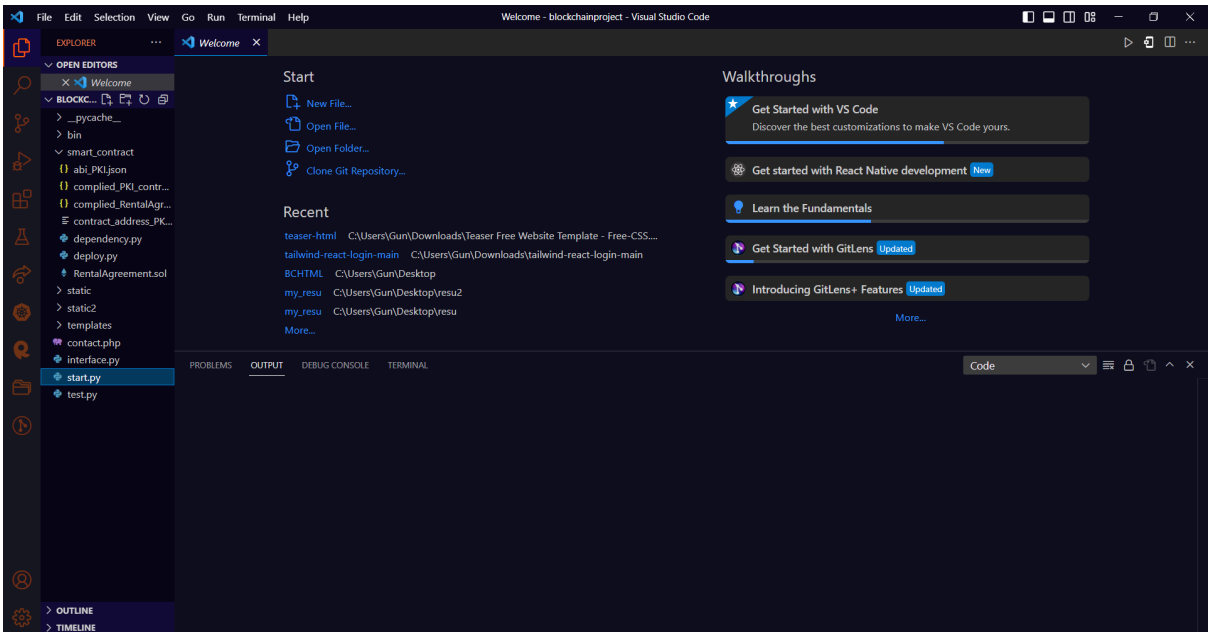
User Interface and User Experience Design

Instructions manual

Please follow all the instructions to use this website correctly. IF there is an pop up error don't be discouraged, click okay and continue. But in the worst case scenario, you have to restart your computer and start this manual again.

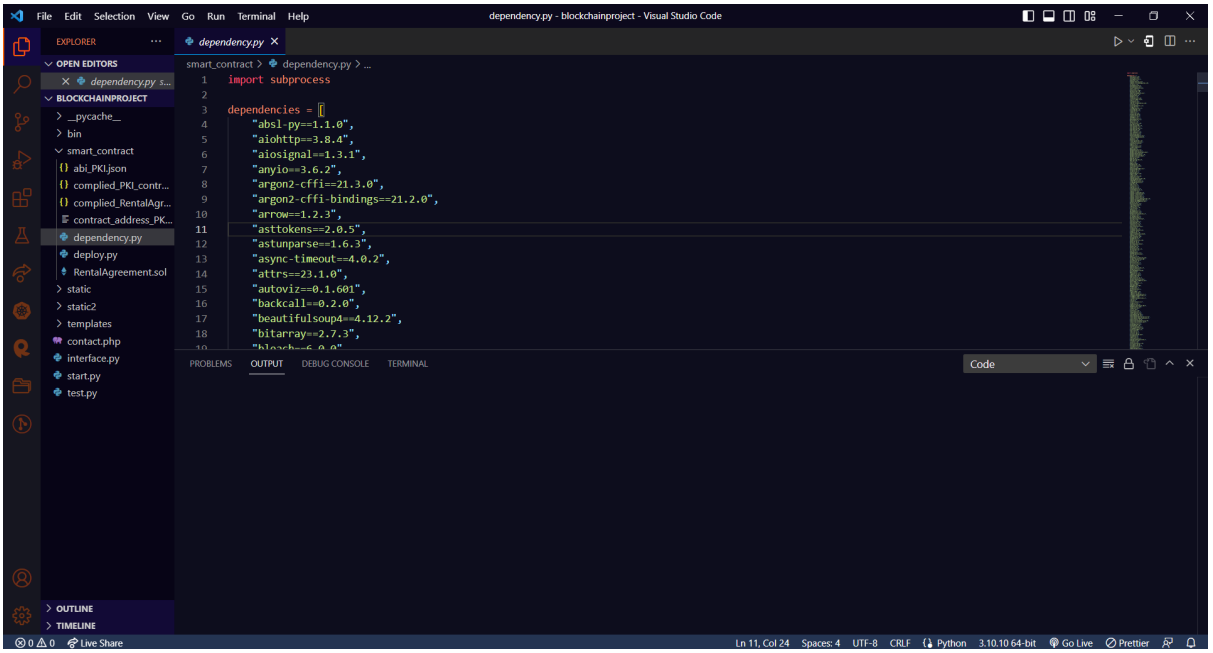
1. Getting started

Open the folder and go to smart_contract folder.



2. Install All Dependencies

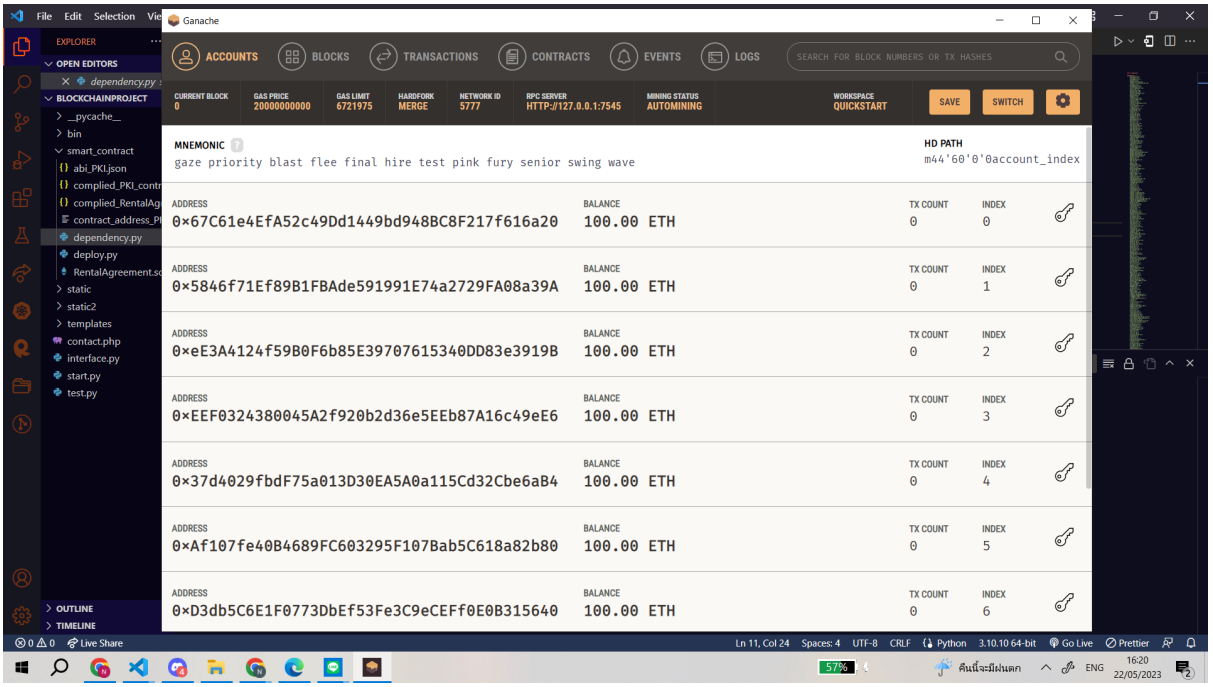
Open dependency.py and run the code to install all dependencies



3. Open Ganache

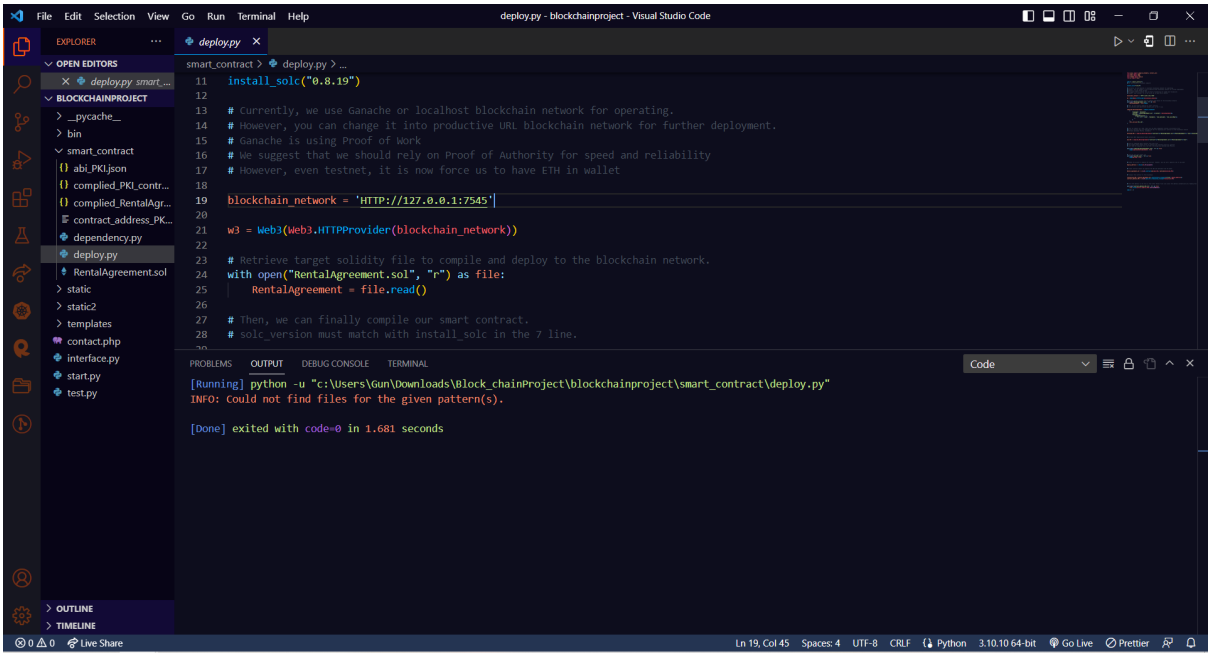
Open and Ganache and quick start with Ethereum

Set your port to **HTTP://127.0.0.1:7545**



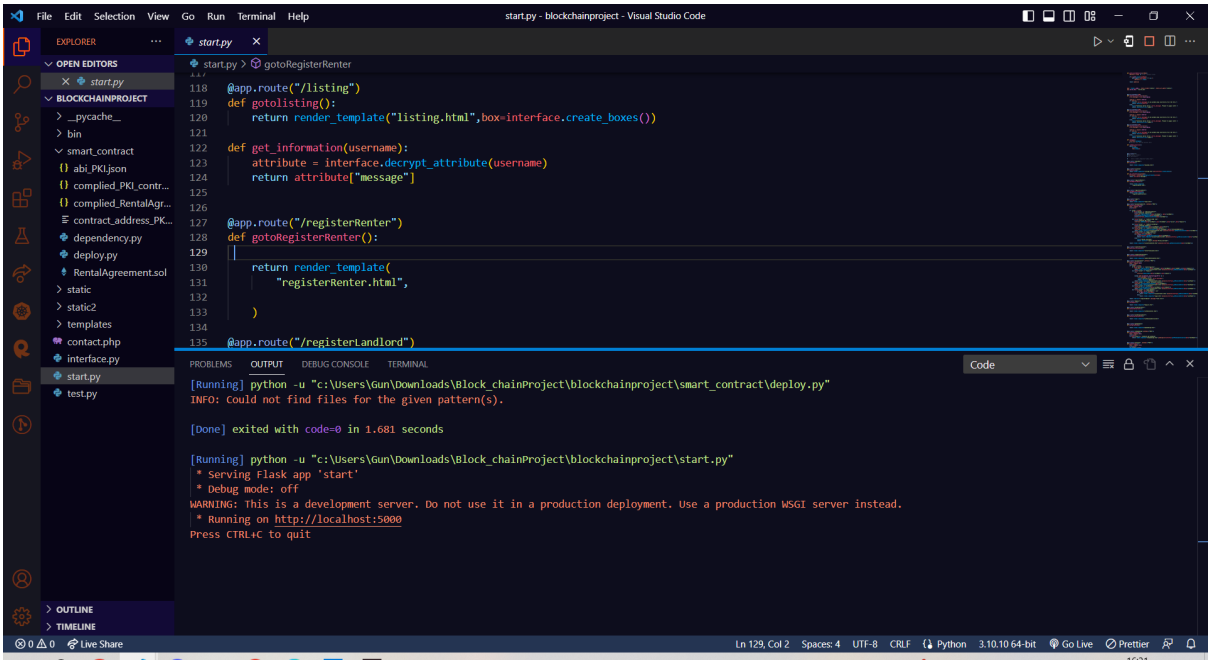
4. Deploy the contract

Go to deploy.py and run the code. It may show these results. but ignore the INFO part.



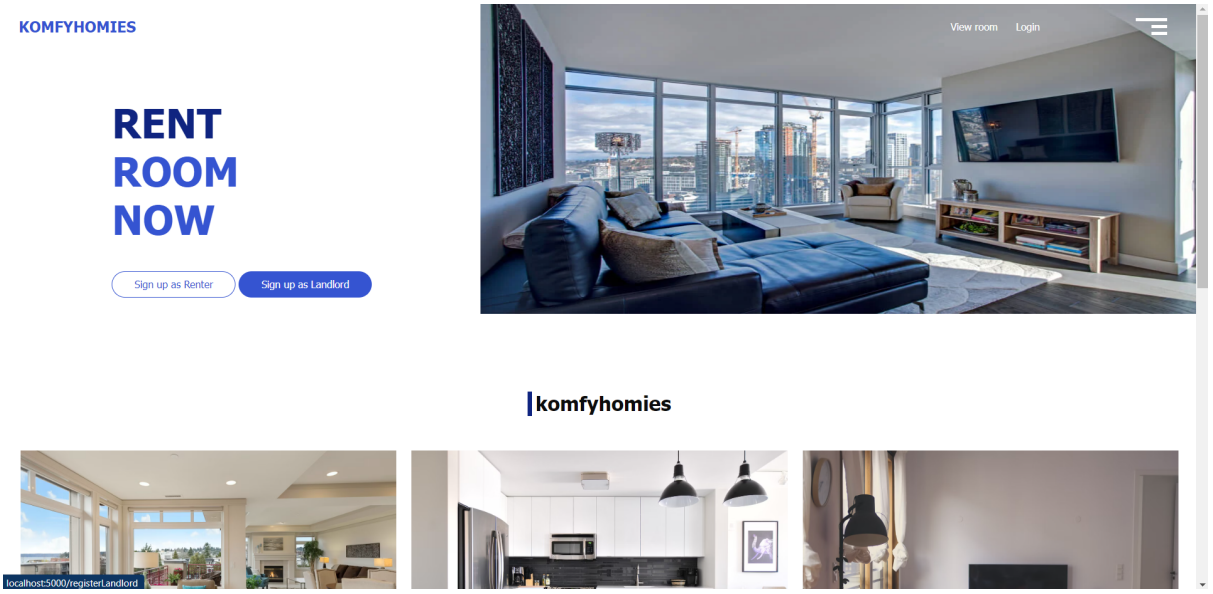
5. Start the website

Go to start.py and run the code. It may show these results. then open in /localhost:5000



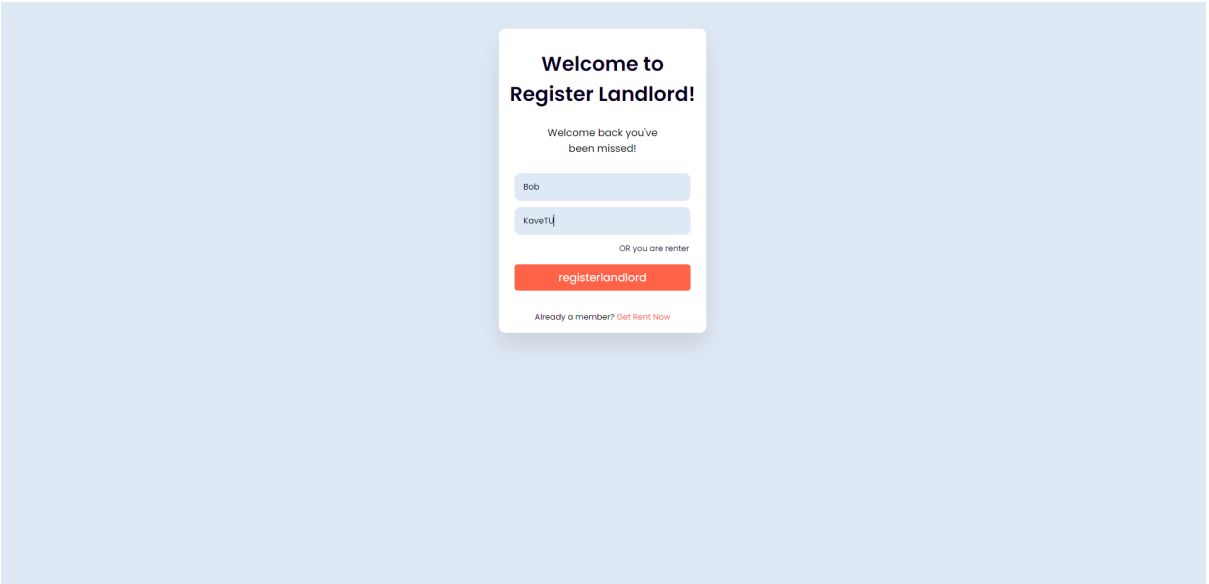
6. Home Page

First, if you want to be a landlord. Please press sign up as Landlord.



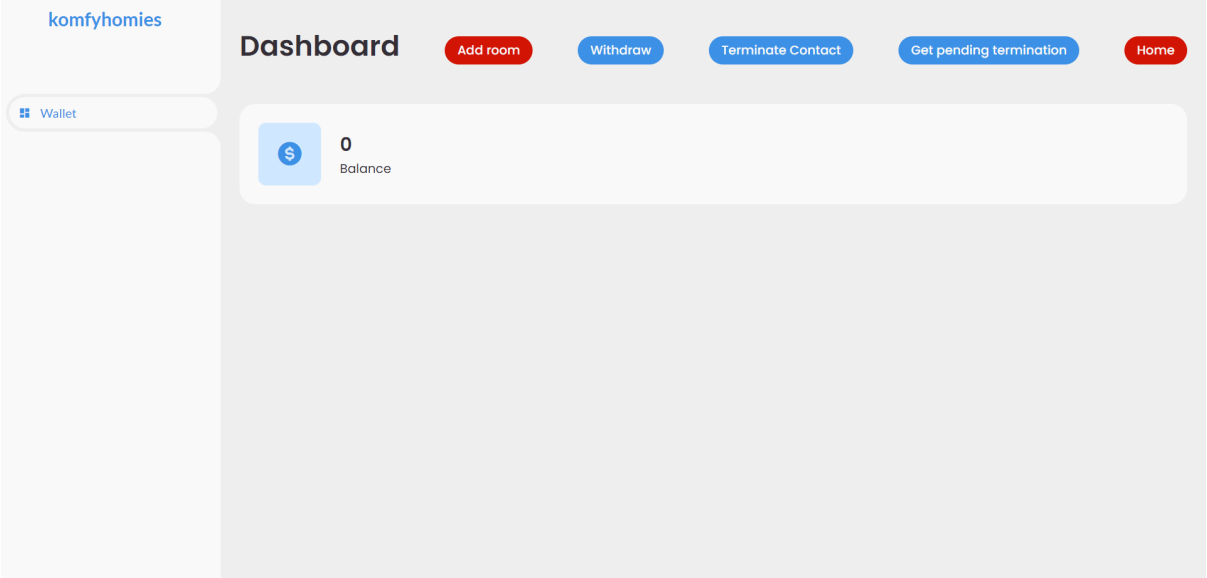
7. Register Landlord Page

Now let's register to be a Landlord, fill Username and Dorm Name, which is your dominitory name. Then press register Landlord.



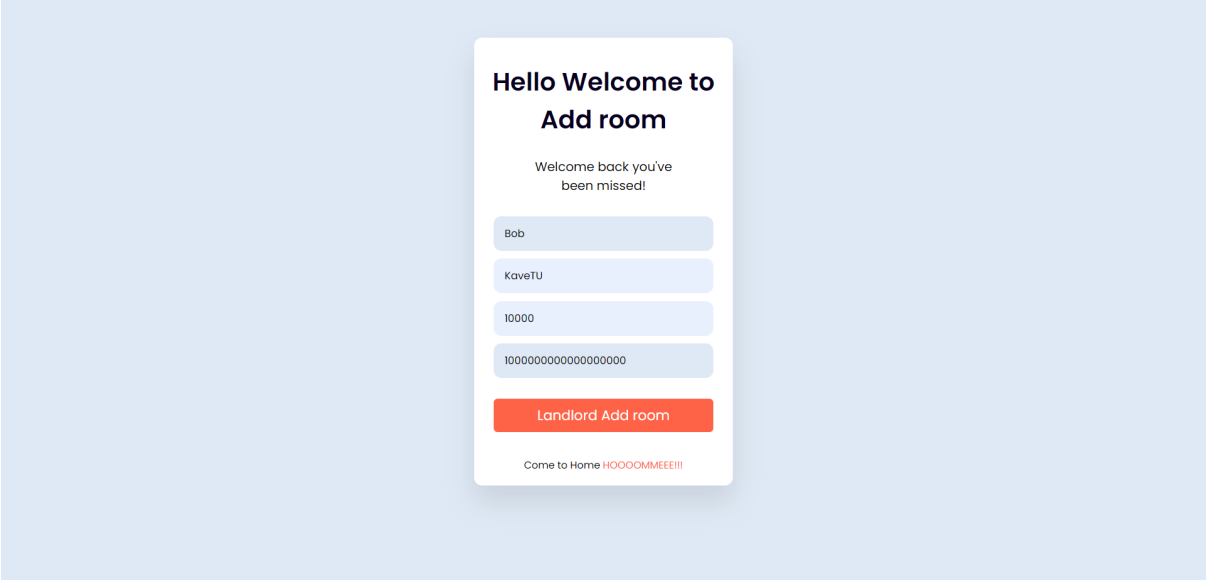
8. Account Landlord Page

After you register to be Landlord, It will lead you to the Landlord Page. Next let's add some rooms, by clicking on the add room button.



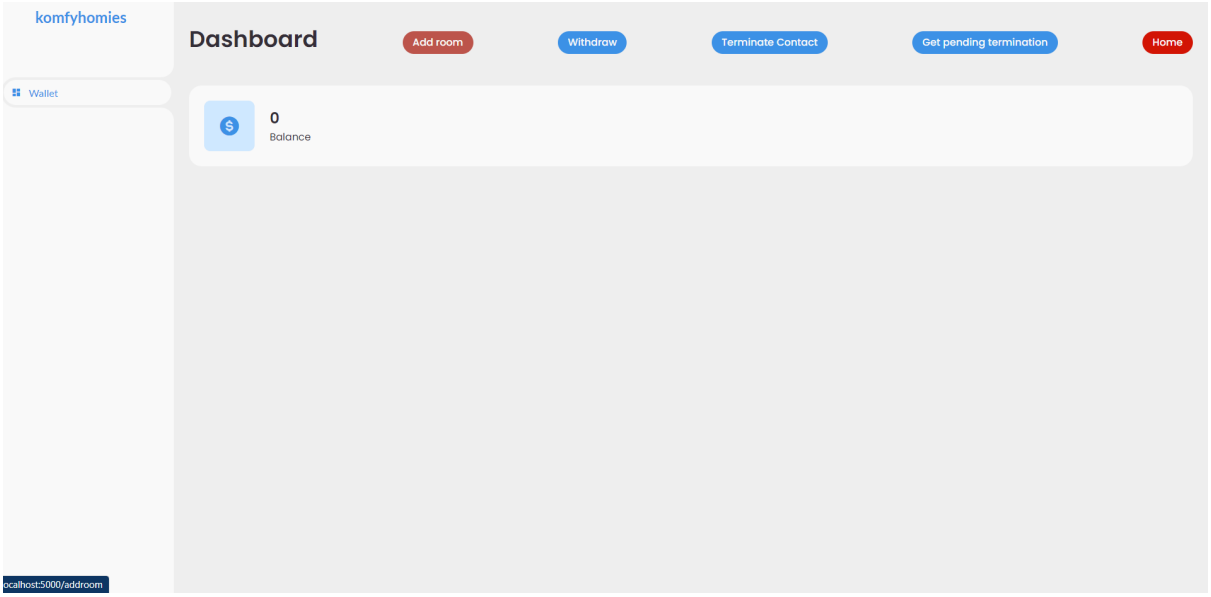
9. Add Room Page

Please fill all of this form, start by filling the username and dorm name that you registered. Then enter the price you want, also the deposit amount that renter have to deposit into this contract.



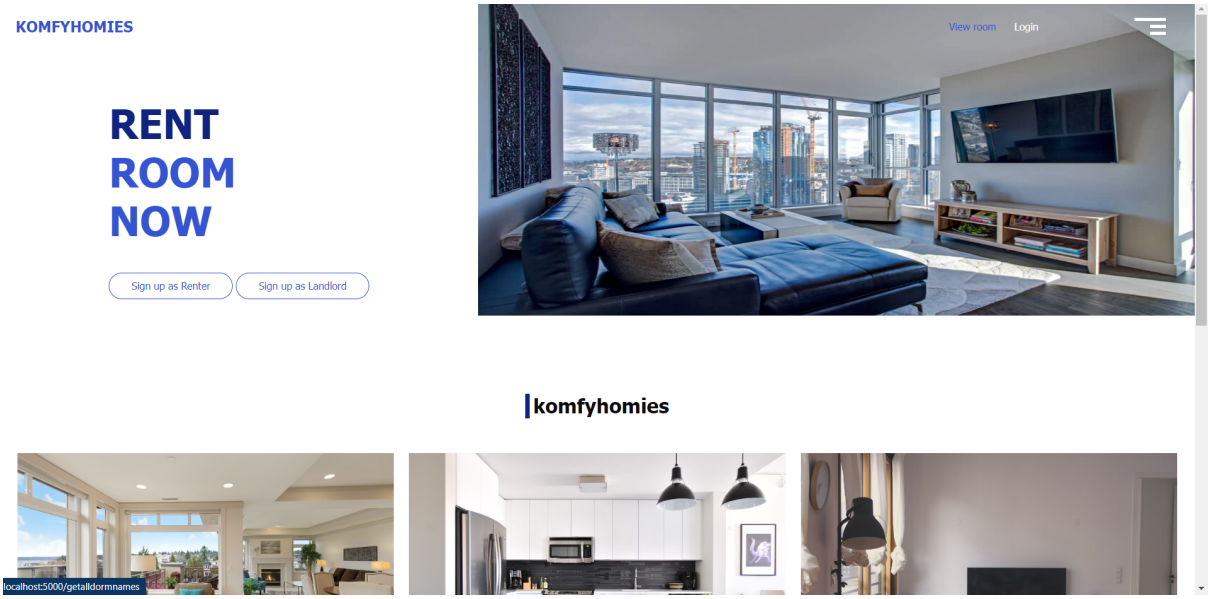
10. Landlord account Page

Ok after you add the room, we come back to this page and now we will try to rent some room with a different username.



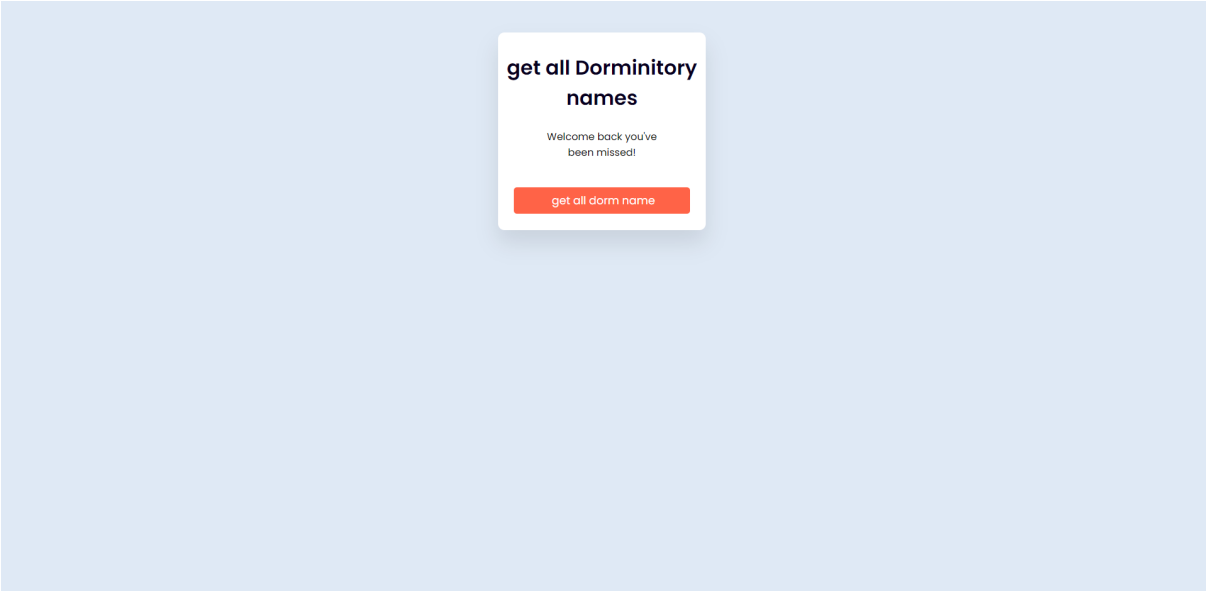
11. Home Page

Now let's check some rooms before renting them. go to the view room in the top right corner.



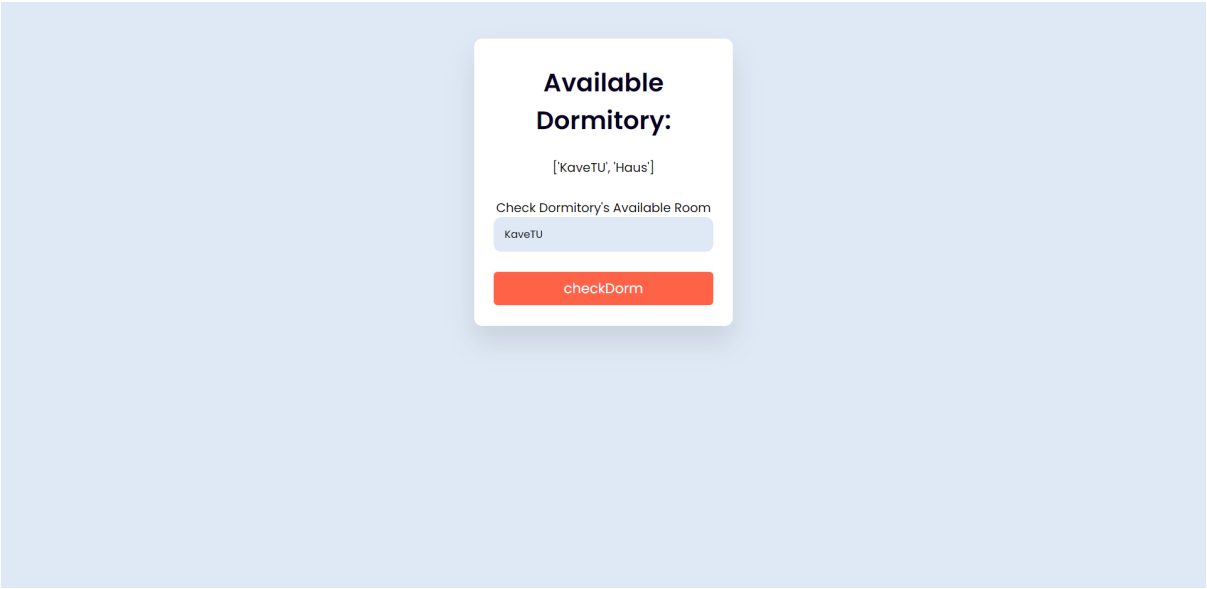
12. Get Dorm name Page

Let's get some names of the dormitories that are available. click get all dorm names.



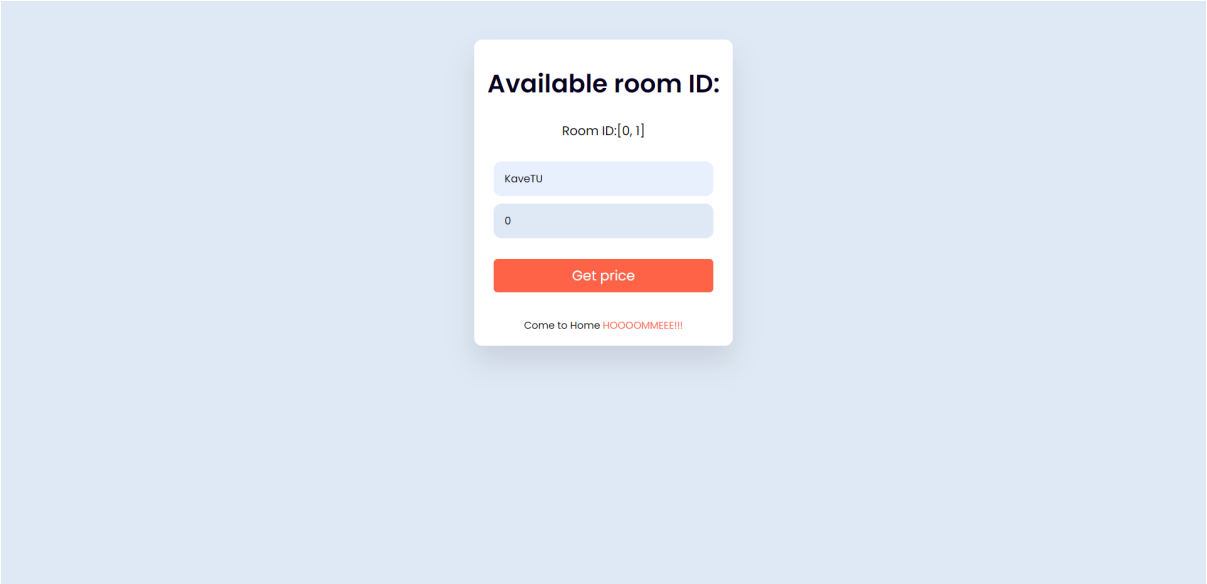
13. Get Available Room Page

Type in the dorm name that you want to check whether there are rooms available or not. For example, Assign KaveTU to check available rooms.



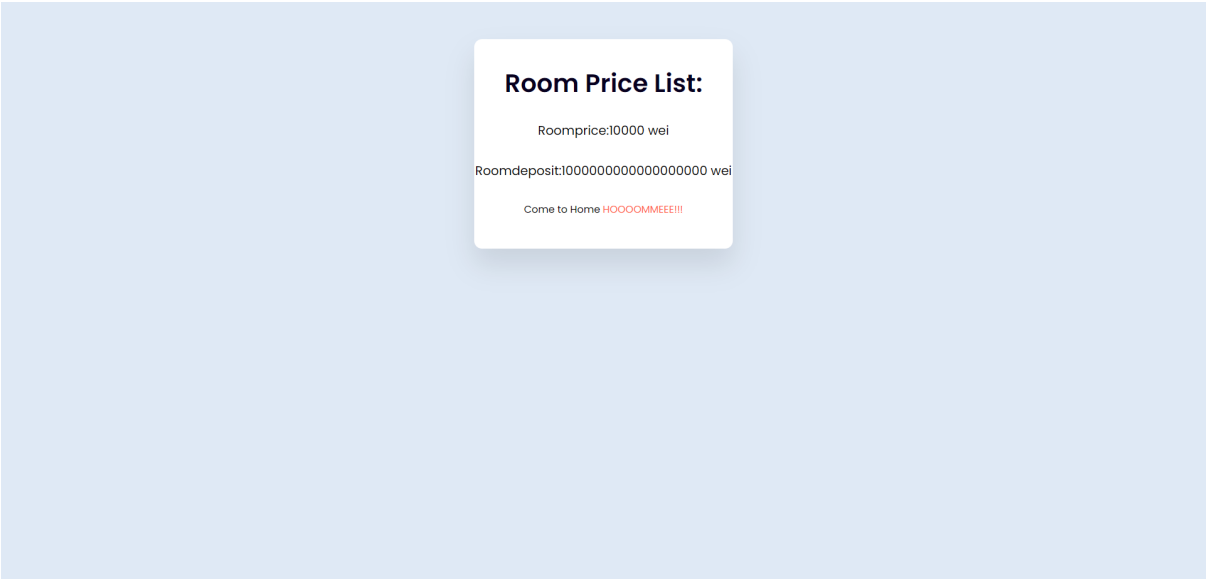
14. Get Room Price Page

In this page we will type in dorm name and roomId to get the price.



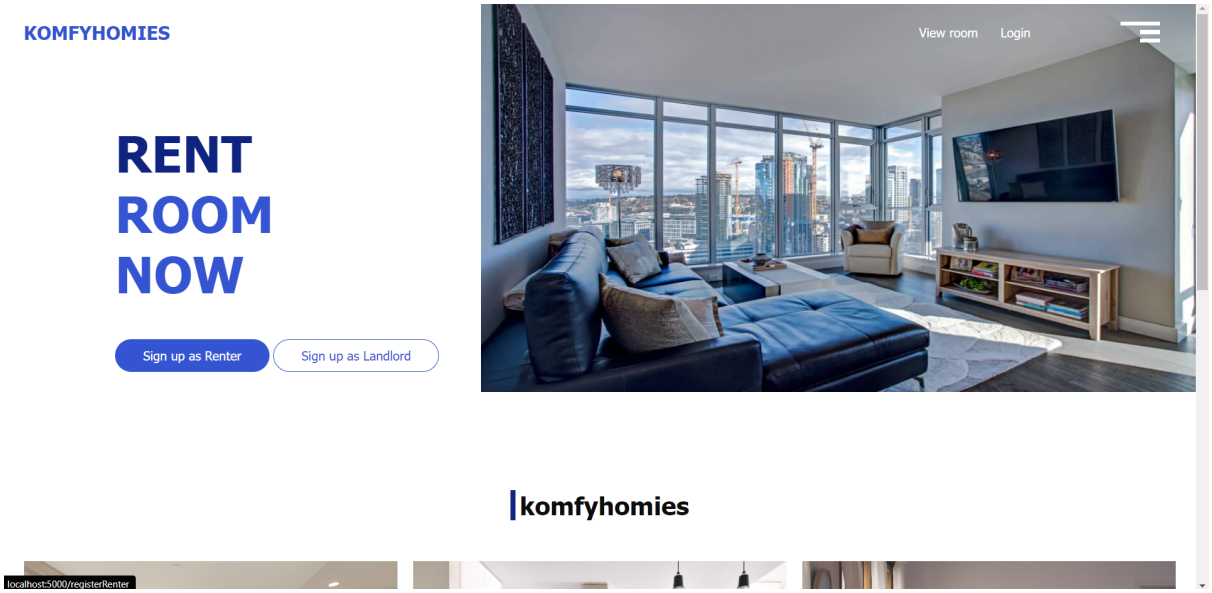
15. Showroom Price Page

In this page we show the room price and how much the deposit. Now let's go back to home by clicking HOOOOOME!



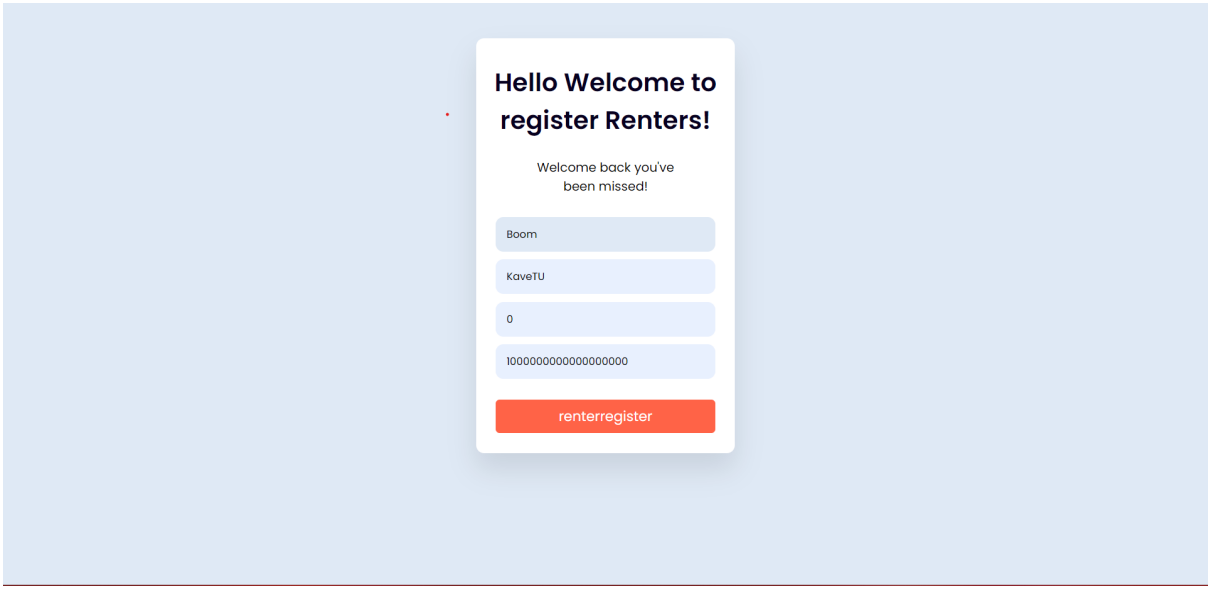
16. Home Page

In this page we show the room price and how much the deposit. Now let's go back to home by clicking HOOOOOME!



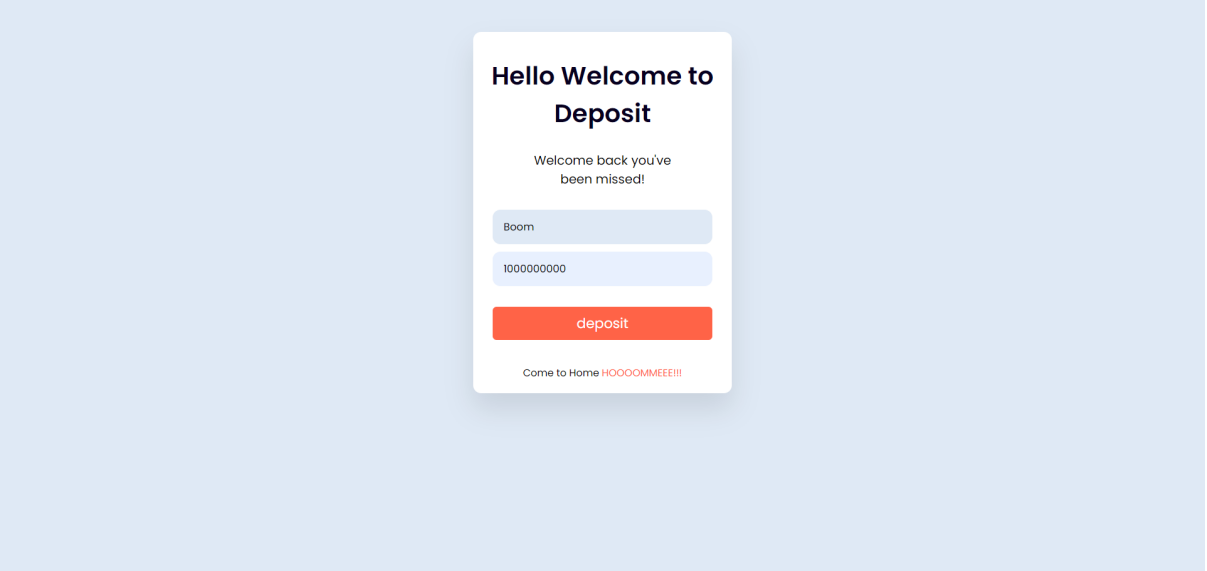
17. Register Renter Page

On this page we will register as renters. First fill the username and dorm name that you want to assign in, also the room Id and the money you have to pay for room deposit.



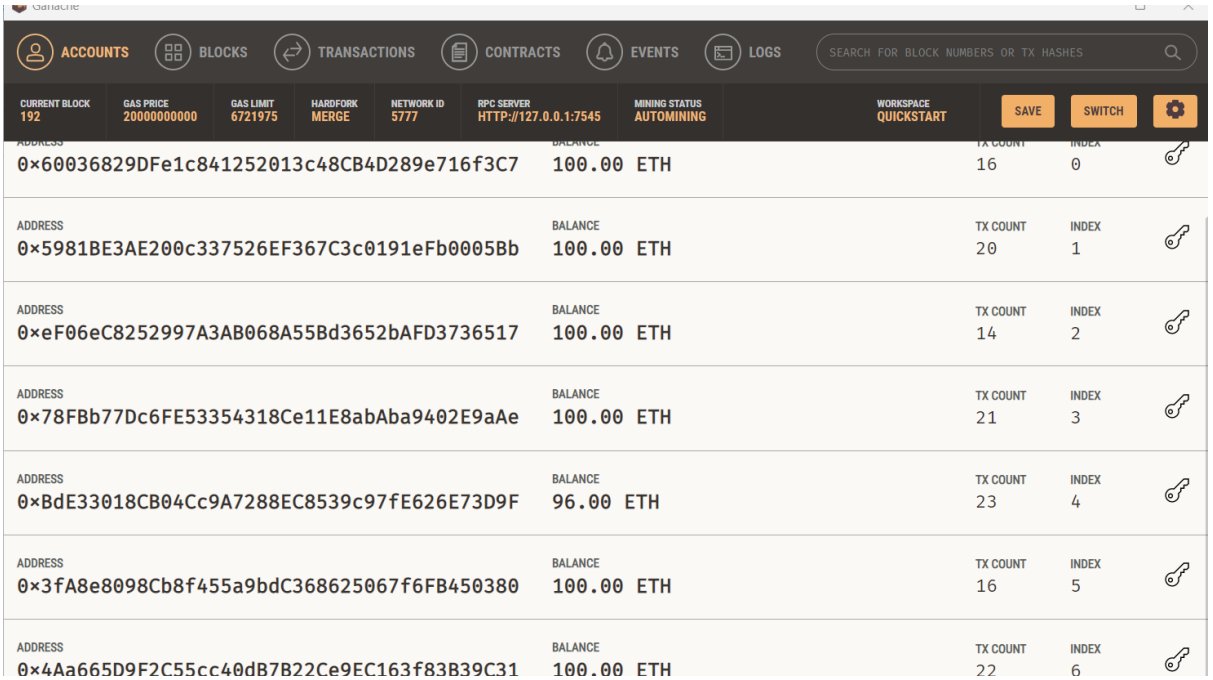
18. Deposit Renter Page

You must **deposit first in 2 minutes** because the money will be cut to landlord every

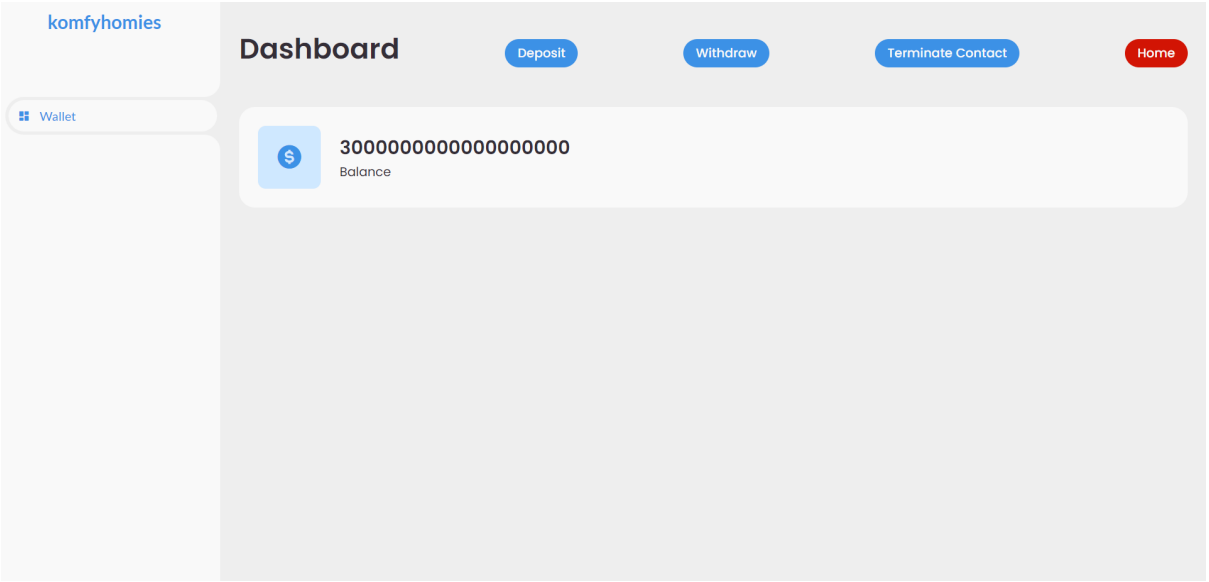


30 seconds else the contract will be terminated

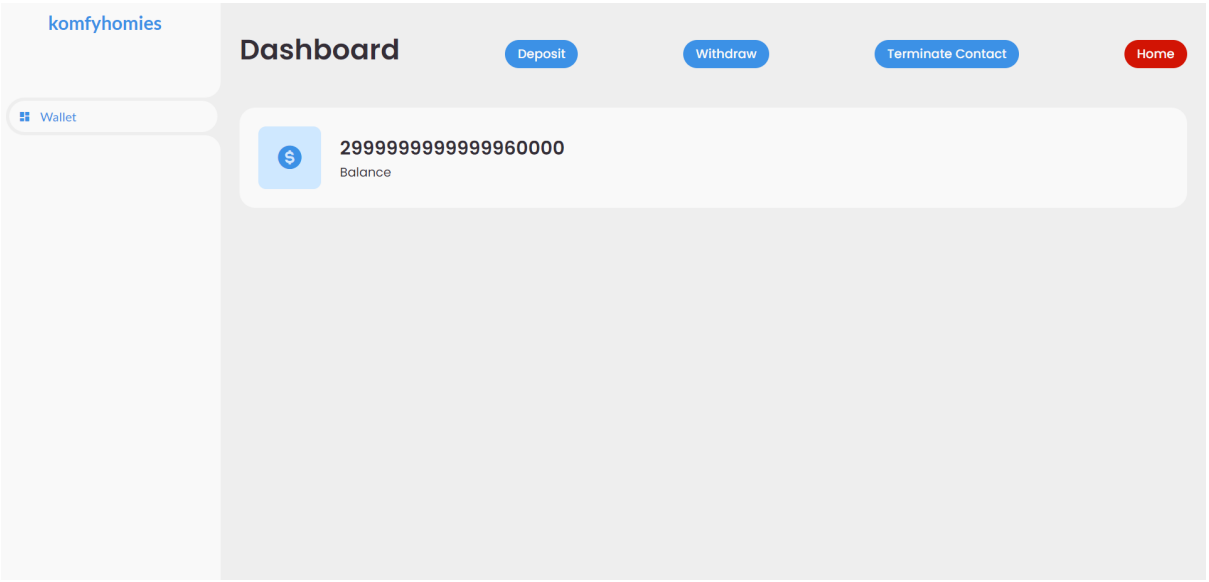
The money have been pulled from the Ganache by 3000000000 wei or 3 eth



19. Account Renter Page



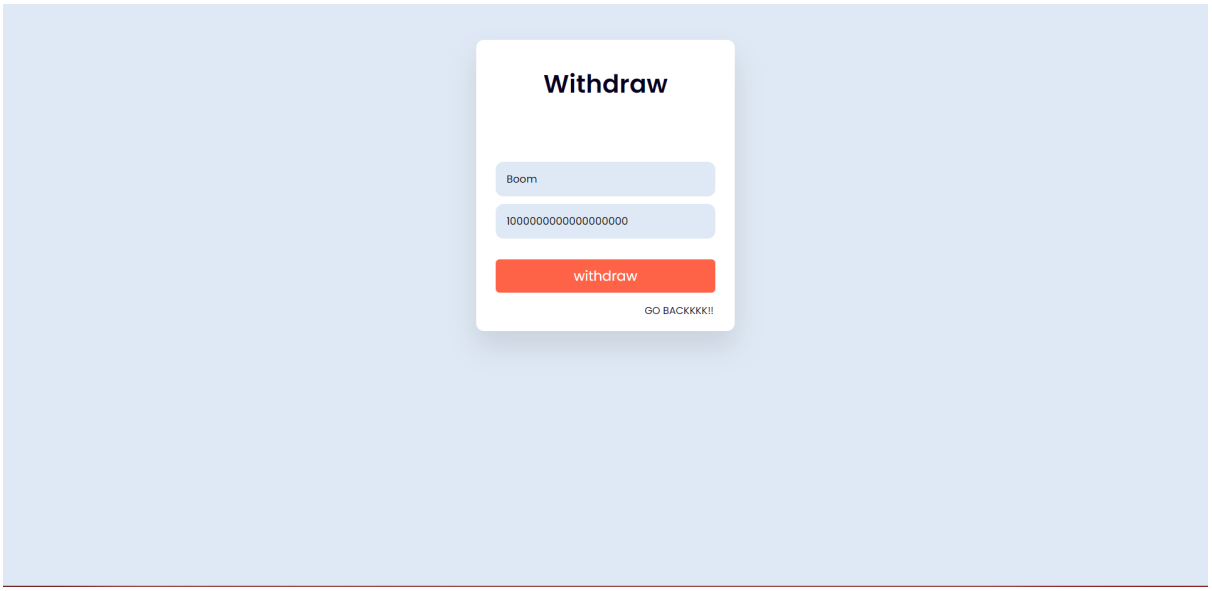
For the sake of testing instead of setting rent due interval from 30 days to 2 minutes as the rent price is 10000 so it will cut 10000 wei every 2 minutes

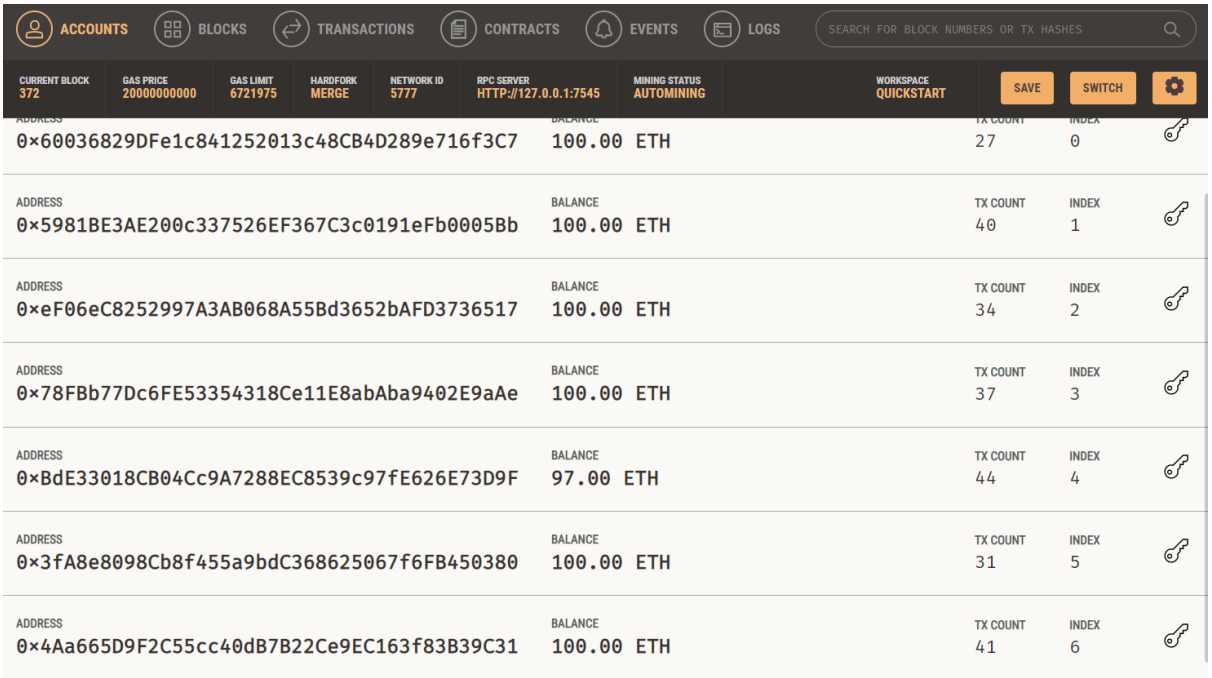


For the sake of the testing we will withdraw 1 eth and the money goes back to the Ganache !

20. Withdraw Page

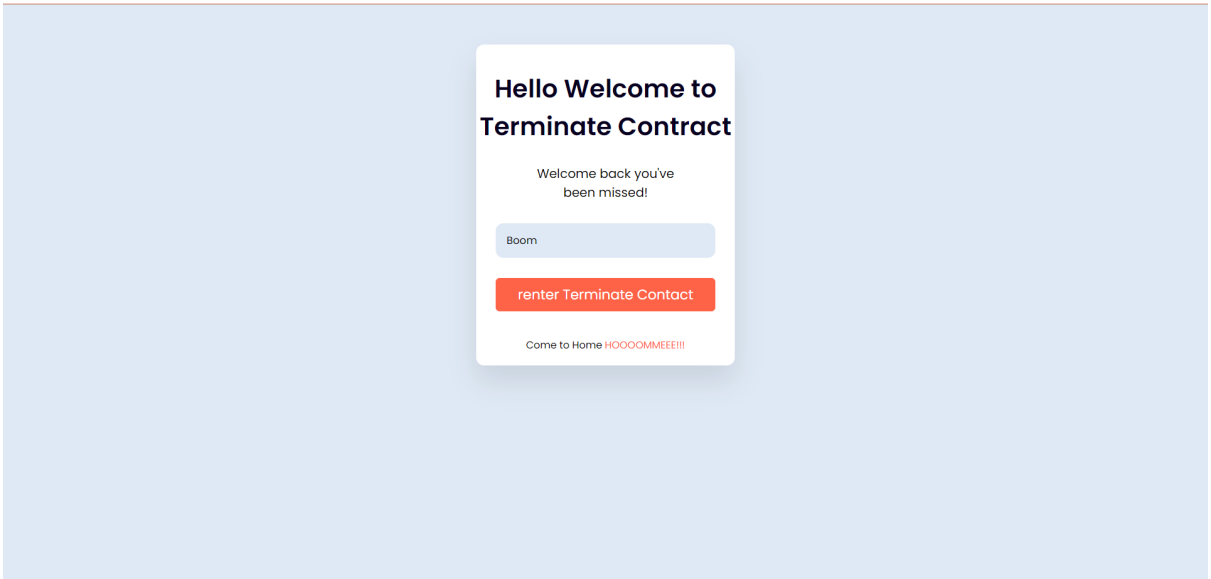
Click the withdraw button on the Account Renter Page Then fill user name and amount you want to withdraw.





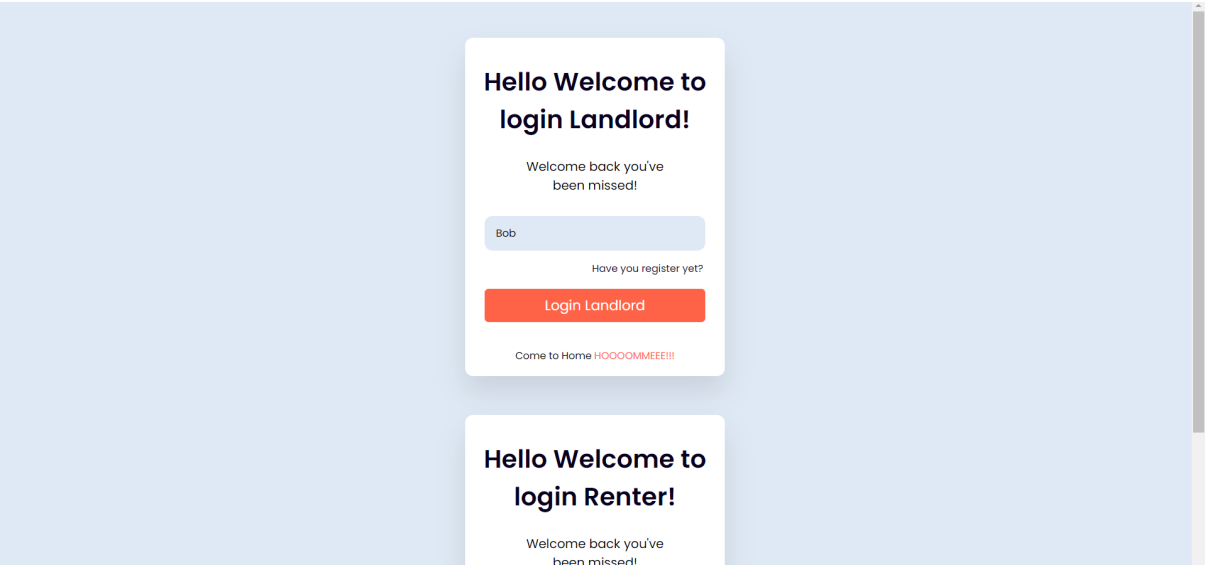
21. Terminate Contract Page

Let’s say the renter want to terminate the contract Clicking the button will get you to the termination pending list



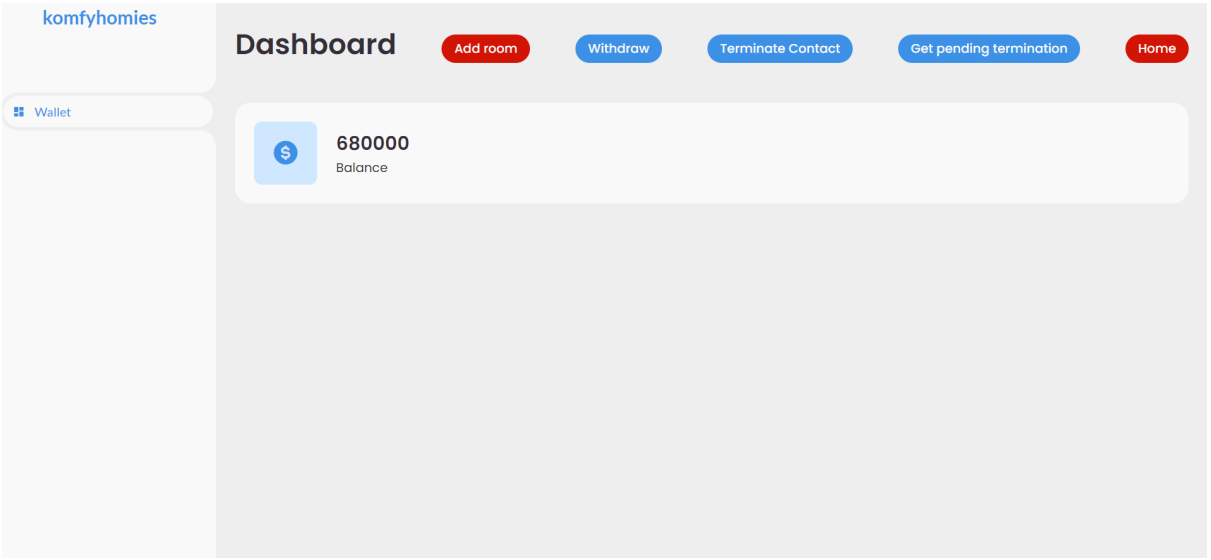
22. Login Page

At this point the landlord already got 680000 from the rent



23. Login Page

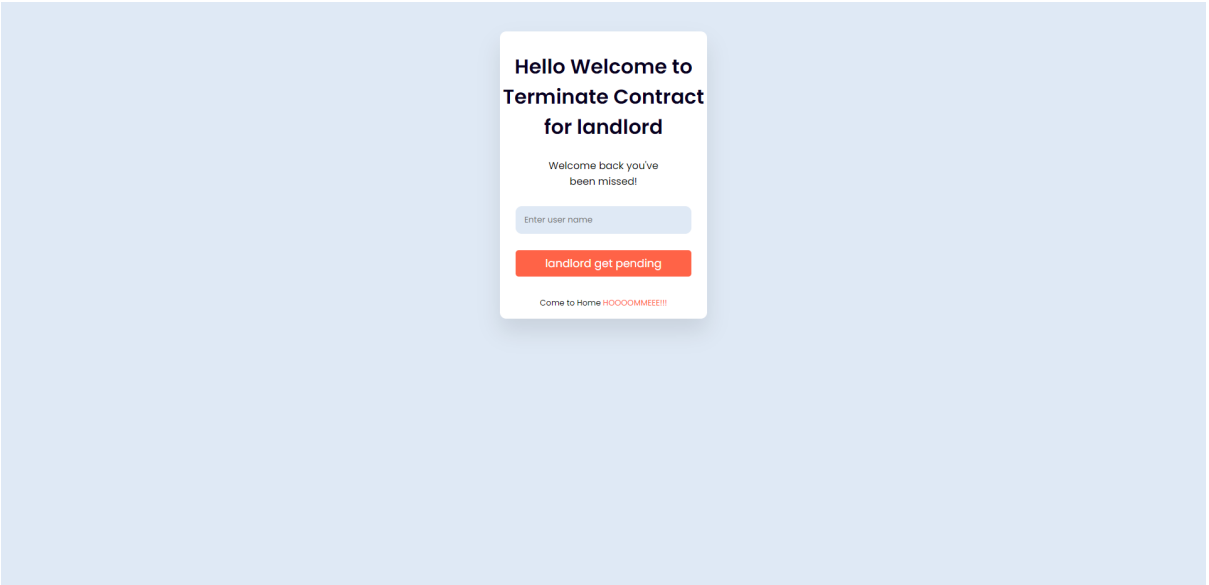
Now we press get pending termination



Put **Landlord** user name whom in our testing is called Bob

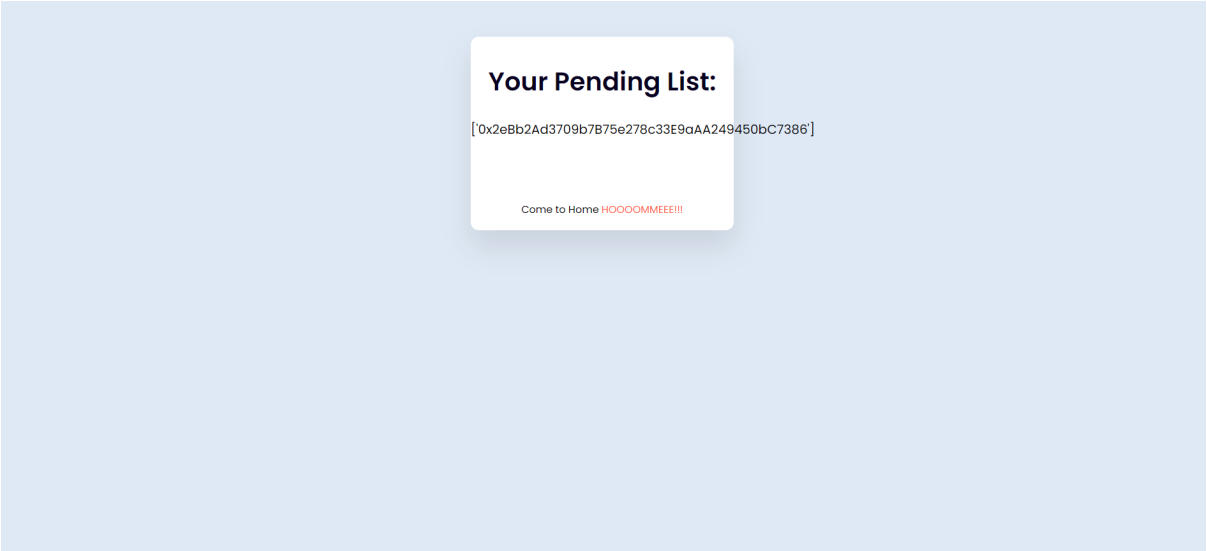
24. Show Pending list Page

then we click to get pending termination to get address of the renter who's want to terminate the contract



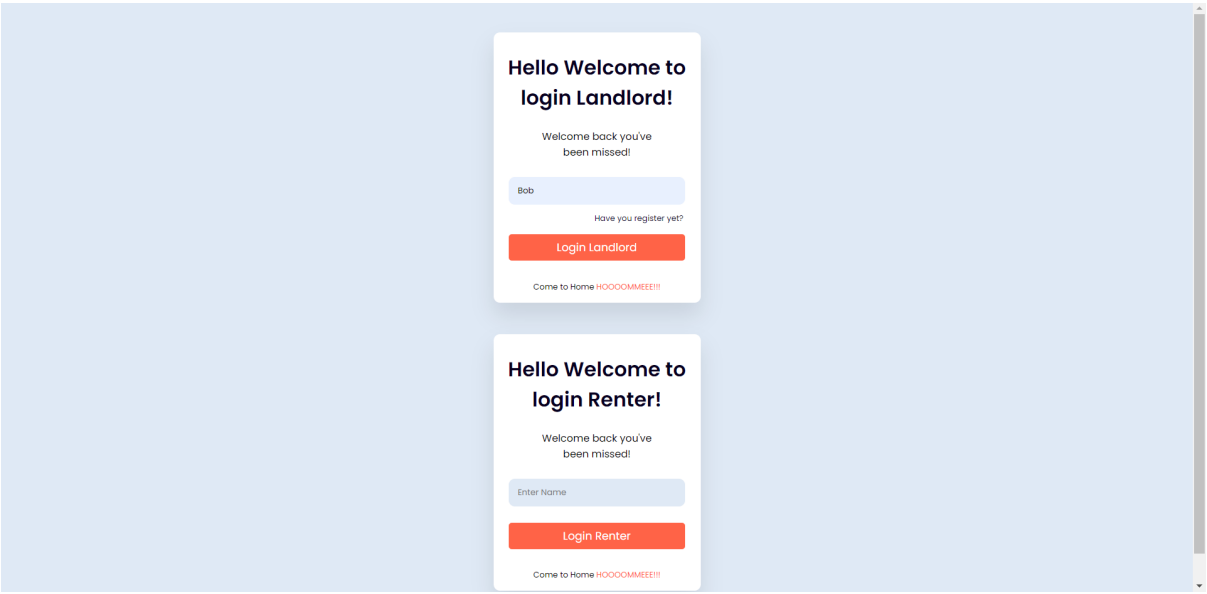
26. Show Pending list Page

then it will show the Pending List which contains the address.



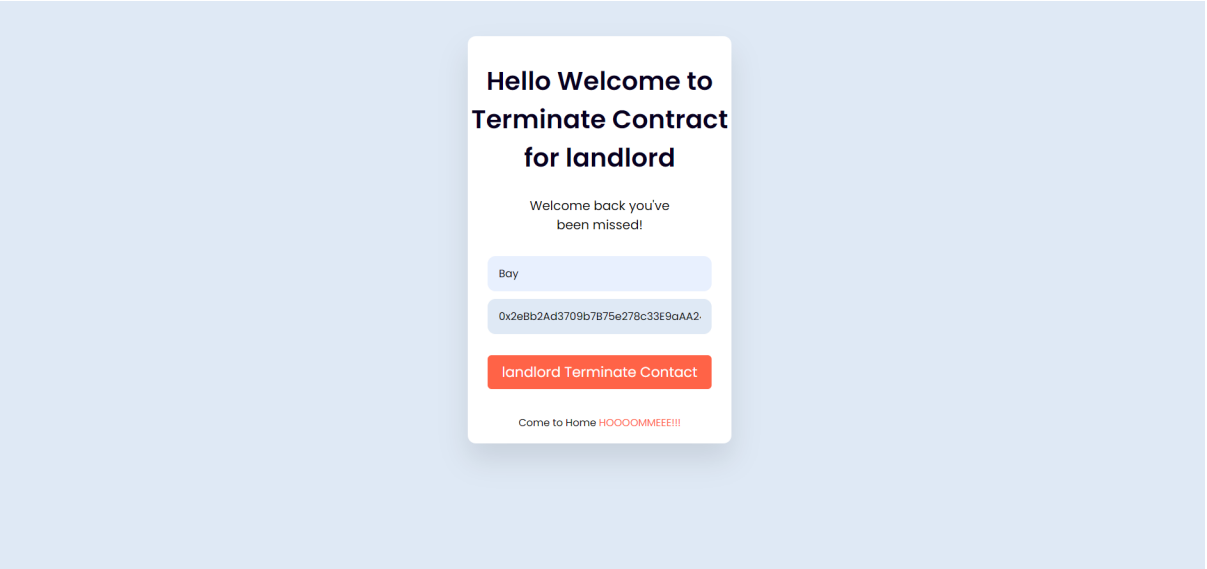
25. Now we go back to the Home page and login as Landlord

login by using your username



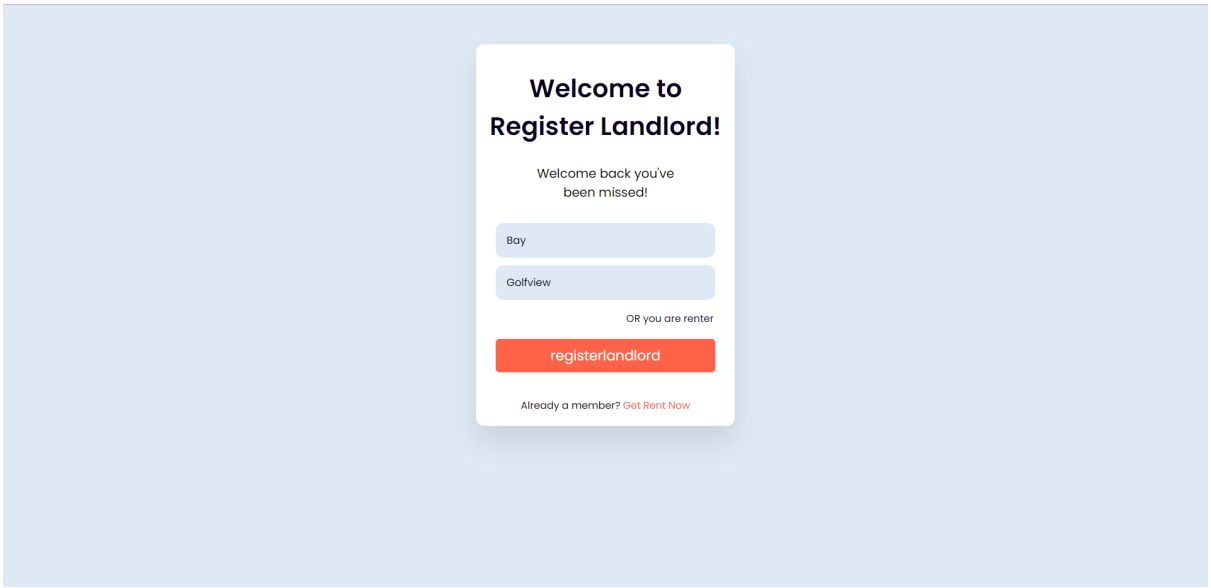
27. Terminate Contract as the Landlord

after you submit this renter won’t be able to login so they have to register again.

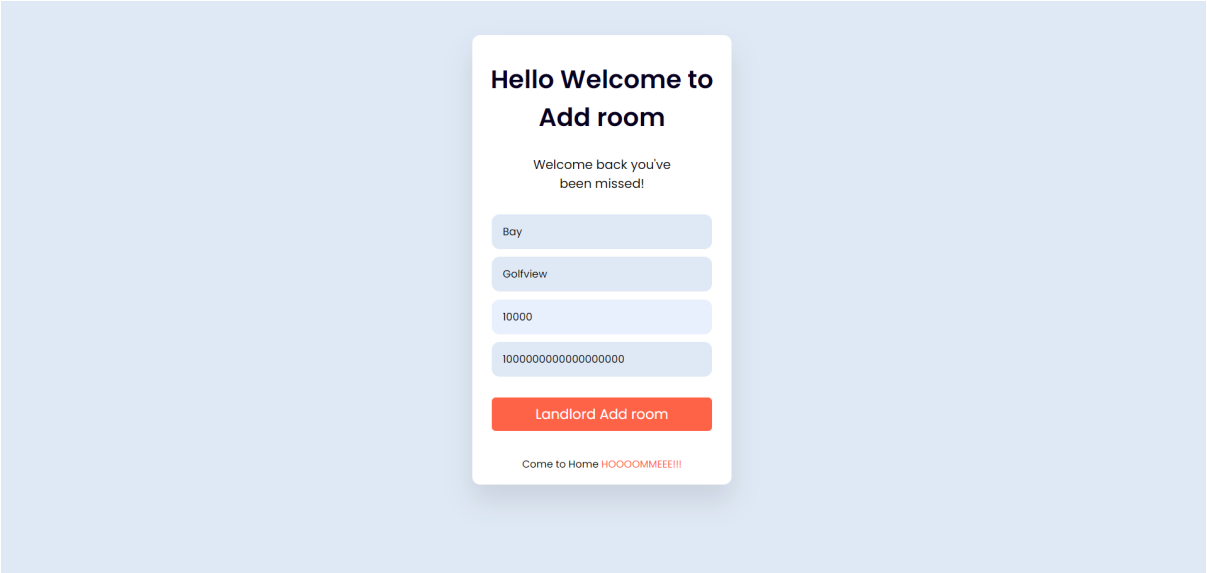


28. Automatically Terminate Contract

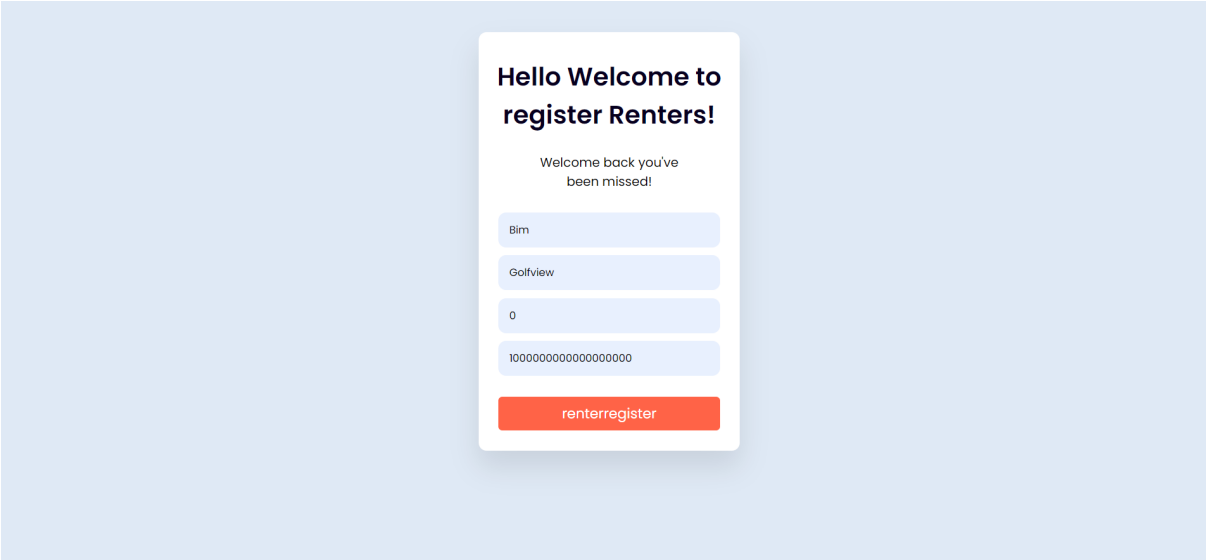
'This page will show you the address of the pending termination renter. Let's just copy the address then press home and go login again. Then press terminate contract in account landlord and enter the user name of the **landlord** and pending address. Now the users boom contract is terminated and won't be able to login. Now the contract will also terminate automatically. Let's see how.



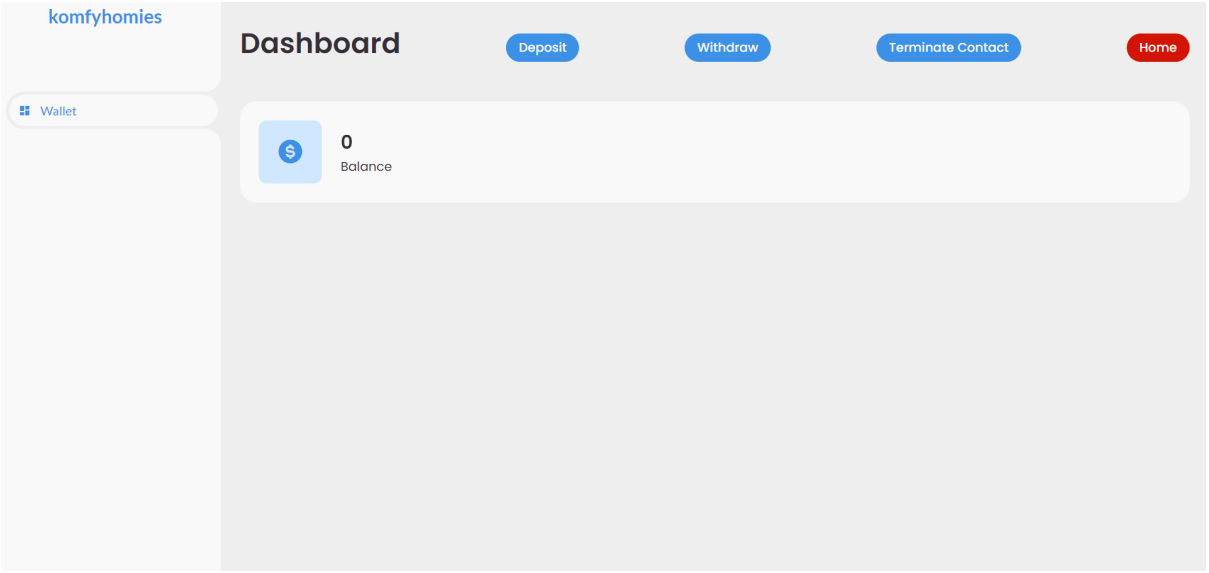
Now let's addroom first

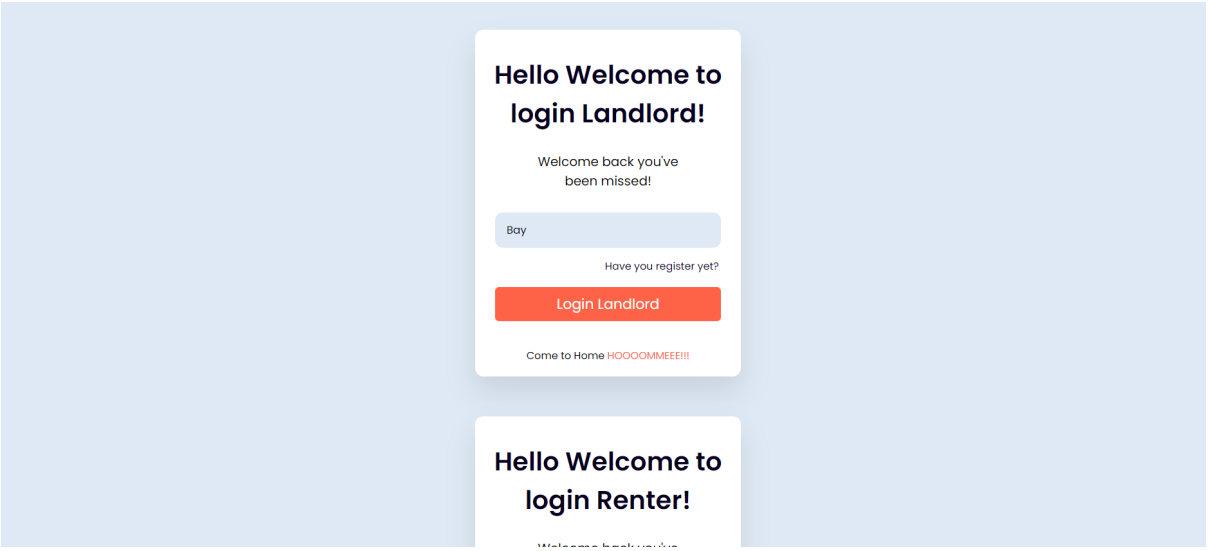


Now lets register as renter

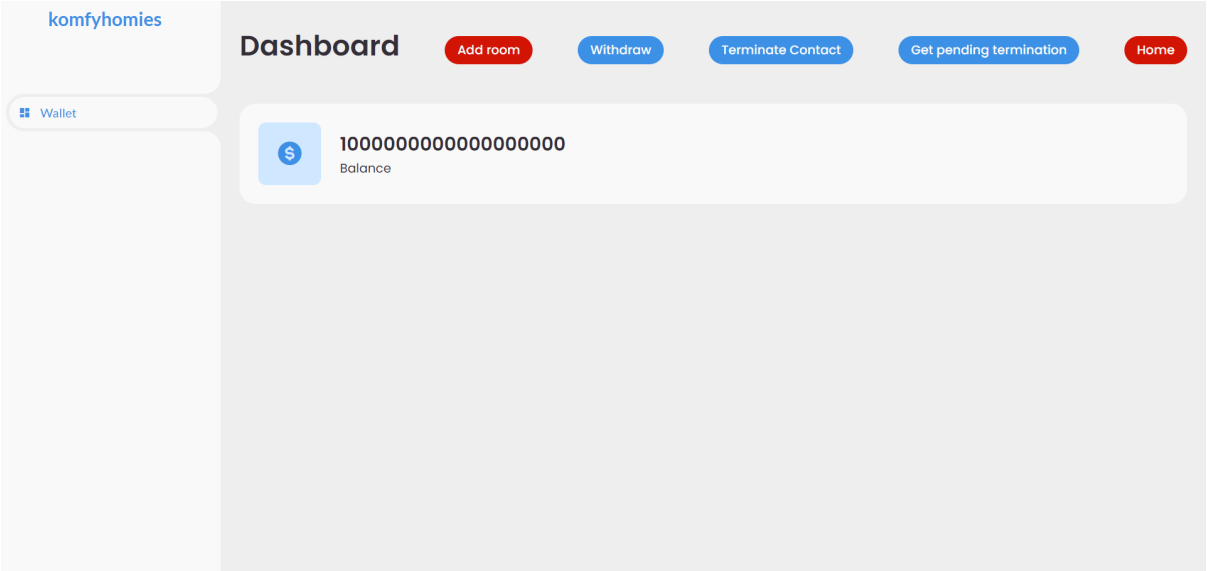


But this time let's not deposit and go to landlord side

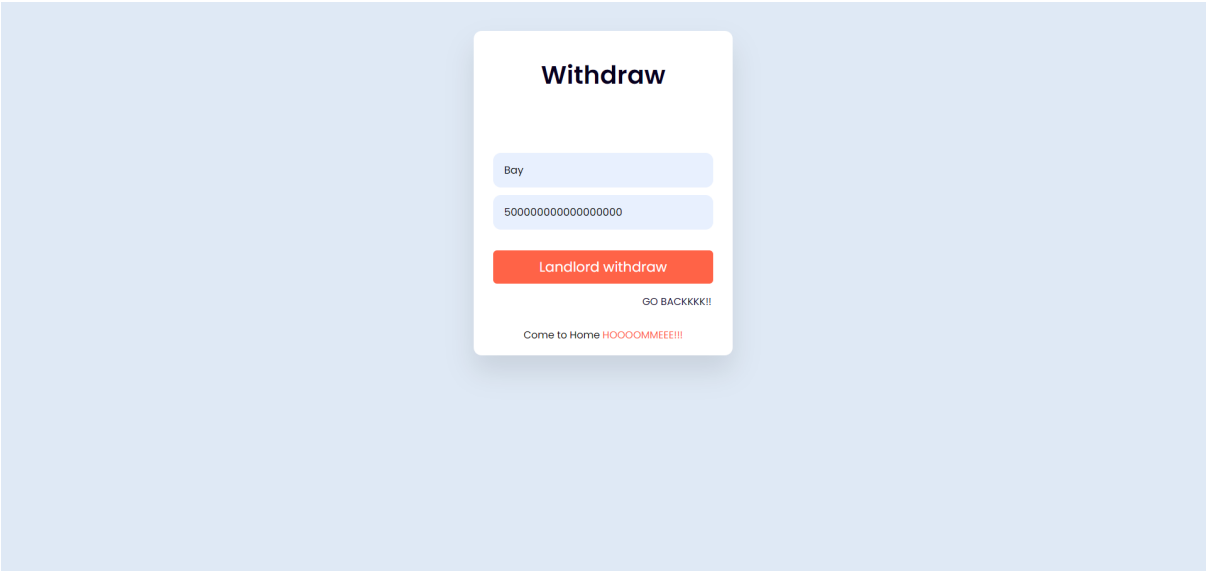




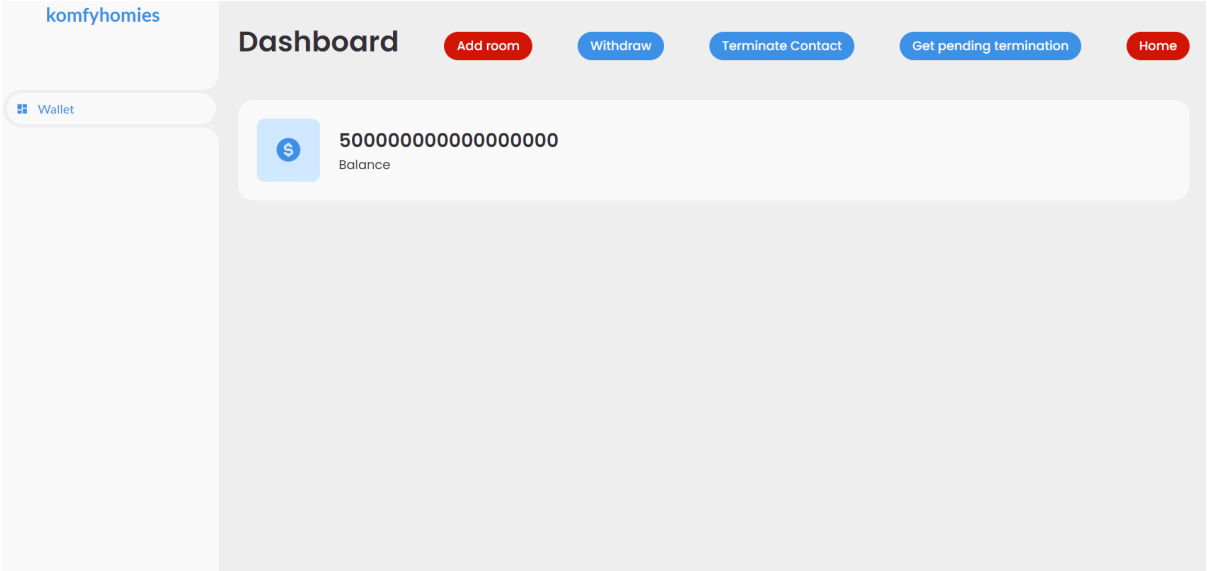
Now you can see that the contract has been terminated and the room deposit has been transferred to the landlord.



Lets withdraw about 0.5 ETH



And the landlord balance get deducted



And the landlord address in ganache balance also increase by 0.5

ADDRESS	BALANCE	TX COUNT	INDEX
0×Bba15D447b75CF0E43DB6C1Fd071A6F84467971A	100.00 ETH	35	1
0×c4F75b9dBfd6560C82943124a986ED801507fd82	100.50 ETH	29	2
0×24F5661dC5E485Cd820402AfD0d6C913CD1C55B4	100.00 ETH	34	3
0×990d8C6c1F1da7310E6e1188E417D428B8B27743	100.00 ETH	32	4

So we have completed all the functions:

- Register Renters
- Register Landlord
- Add room
- Deposit for renter
- Termination page on both side
- Auto termination
- Withdraw for both landlord and renters

\

Chapter 4

Conclusion and Discussion

Conclusion

The proposed platform thus satisfies the criteria for user registration, dormitory management, room availability and booking, rent payments and deposits, rental agreement management, termination and dispute resolution, security and privacy, user interface and experience, as well as testing and quality assurance.

The frontend, the backend, and the smart contracts make up the system architecture's three main parts. The platform's frontend, which was created using HTML/CSS, offers a user-friendly interface for landlords and tenants to interact with. The deployed Solidity smart contracts in the backend interact with the local blockchain network and handle communication with the frontend. The backend is powered by Python, Ganache, and Flask.

In conclusion, the suggested platform provides a secure, open, and decentralized system for managing dorms, renting out rooms, handling rent payments, and managing rental contracts for both landlords and tenants.

Obtained Benefits

- Understand more about developing decentralized applications and how it works
- Understand more about the blockchain
- Be able to learn to create web3.0 but it is quite really hard for us to do it within a month with non knowledge about it.

Performance and Limitations

- **Gas Fee and ETH:** Gas fees are an essential aspect of the Ethereum blockchain network. They serve as transaction fees and are paid by users to execute smart contracts and interact with decentralized applications (dApps) on the network. Gas fees ensure that computational resources are allocated efficiently and prevent malicious actors from flooding the network with spam transactions.When deploying and maintaining a blockchain project with an economic purpose, it is crucial to consider gas fees. Gas fees directly impact user experience and can affect the feasibility and adoption of a project. High gas fees can discourage users from engaging with dApps, especially for smaller transactions or activities with lower economic value. To mitigate the impact of gas fees, developers can employ various strategies. One approach is to optimize smart contracts and reduce the complexity of operations, which can lower gas consumption. Additionally, developers can implement off-chain scaling solutions like layer-2 protocols or sidechains to reduce reliance on the Ethereum mainnet for every transaction. These solutions help mitigate the impact of high gas fees and provide a smoother user experience.

Security Issues:

- **Weak Input Form Validation:** Weak input form validation refers to a situation where the validation process of an input form is insufficient or incomplete, leaving potential vulnerabilities in the system. Proper input form validation is crucial to ensure data integrity and protect against various security risks, such as SQL injection, cross-site scripting (XSS), and code injection attacks. To improve input form validation, it is important to implement robust validation mechanisms that thoroughly check and sanitize user input. This includes validating the type, length, format, and range of input data. Additionally, input should be properly encoded or sanitized to prevent malicious code injection or XSS attacks.
- **Insecure Password Submission:** Using a username as a password or employing weak password submission methods can compromise the security of a system. To enhance password security, it is important to follow established best practices:
- **Password Complexity:** Encourage users to create strong passwords by setting requirements for complexity, including a mix of uppercase and lowercase letters,

numbers, and special characters. Avoid allowing common passwords or easily guessable patterns.

- **Password Encryption:** Store passwords securely by using strong encryption algorithms, such as bcrypt or Argon2. Never store passwords in plain text or reversible encryption formats.
- **Two-Factor Authentication (2FA):** Implement 2FA to add an extra layer of security. This can involve using SMS verification, authenticator apps, or hardware tokens to verify the user's identity.
- **Password Policies:** Enforce periodic password changes and avoid password reuse. Educate users about the importance of strong passwords and provide guidelines for password management.
- **Password Hashing:** Use salted hashes to further enhance password security. Salting involves adding a unique random value to each password before hashing it, making it more resistant to precomputed attacks.

Chapter 5

References

- Qi-Long, C., Rong-Hua, Y., & Fei-Long, L. (2019). A Blockchain-based Housing Rental System. In Proceedings of the International Conference on Advances in Computer Technology, Information Science and Communications (CTISC 2019), pp. 184-190. ISBN: 978-989-758-357-5. DOI: 10.5220/0008097201840190.
- B. K. Mohanta, S. S. Panda and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bengaluru, India, 2018, pp. 1-4, doi: 10.1109/ICCCNT.2018.8494045.
- Karamitsos, I. , Papadaki, M. and Barghuthi, N. (2018) Design of the Blockchain Smart Contract: A Use Case for Real Estate. *Journal of Information Security*, **9**, 177-190. doi: 10.4236/jis.2018.93013.
- Z. Zhu, J. Su, Z. Jiang, M. Ye and Z. Zheng, "Making Smart Contract Classification Easier and More Effective," *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, Espoo, Finland, 2022, pp. 228-230, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00067.