SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION TECHNOLOGY

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

Final Report

Chainshare

Group Members :

6322772953 Kritsakorn Sripuritanont

6322775080 Tin Wongcharoo

6322772292 Kotchaporn Supakandechakul

Present to

Dr.Watthanasak Jeamwatthanachai

DES484 Topics in Software Technology I

Topic in Blockchain Development

Semester 1 Academic Year 2023 Digital Engineering(DE)

Table of contents

List of figures

Overview and Background

Chainshare(CSHR) is a decentralized application utilizing smart contracts for borrowing real-world items that have gained popularity due to the efficiency and security it offers. This dApp operates on blockchain technology, enabling peer-to-peer transactions without the need for intermediaries like banks or traditional lending institutions.

Overview

1. Concept : Chainshare facilitates borrowing assets in a decentralized manner. Users can lend or borrow items by using a trustless system governed by smart contracts.
2. Blockchain & Smart Contracts : It leverages blockchain's immutable ledger and smart contracts. These contracts enforce terms and conditions agreed upon by borrowers and lenders, automating transactions and ensuring trust between parties without relying on central authority.
3. User Interface : dApp typically offers an intuitive user interface for browsing available items, setting borrowing terms and facilitating transactions. Users can see a profile which indicates status of borrowing, report, listing.
4. Trust and Security : The decentralized nature of these applications ensures transparency and security. Smart contracts execute lending agreements without the need for intermediaries, reducing the risk of fraud or disputes.

Background

1. Rise of Sharing Economy : The sharing economy has seen significant growth, with platforms like Airbnb and Uber transforming how people share resources. Decentralized borrowing apps extend this concept further by removing centralized control and offering a trustless environment.
2. Blockchain's Role: Blockchain's inherent security and transparency make it an ideal technology for peer-to-peer lending and borrowing. Smart contracts enable the automation of lending processes, reducing administrative burdens and enhancing security.
3. Use Cases: These dApps find applications in various scenarios. For instance, someone might borrow power tools for a DIY project, rent out their bicycle for a fee, or offer tutoring services, all managed and secured through the decentralized app's smart contracts.
4. Challenges: Despite their potential, these dApps face challenges such as scalability, user adoption, and regulatory compliance. Scalability issues within blockchain networks might limit the speed and volume of transactions, while regulatory uncertainty might affect widespread adoption.

Case study selection

Case Study: Tool Sharing dApp

Overview

- Concept : The dApp aims to create a peer-to-peer platform for individuals to lend and borrow tools and equipment.
- Target Users : Users who want to borrow things.
- Features : User profile, tool listings, borrowing terms, smart contract-based transactions

Background

- Problem: Many people own tools they rarely use, while others need specific tools for short-term projects but find it inconvenient or costly to buy them.
- Solution: The dApp provides a solution by allowing users to list their tools for lending and enabling others to borrow these tools for a specified duration.

Implementation

- User Interface: A user-friendly interface allowing users to create profiles, list tools, set borrowing terms (duration, fees), and search for available tools.
- Smart Contracts: Smart contracts manage the borrowing process, outlining terms and conditions agreed upon by both parties. They automate transactions and ensure secure lending without centralized control.
- Rating System: Users can rate and review their borrowing or lending experiences, fostering a trustworthy community within the platform.

Case Scenario

- User A: Jane, a homeowner, needs a power drill for a weekend DIY project but doesn't want to buy one for a one-time use.
- User B: John, a hobbyist, has a power drill that he rarely uses and decides to list it on the dApp for lending.

Process

- Jane logs into the dApp, searches for a power drill, and finds John's listing.
- She reviews the drill's details, including borrowing terms, fees, and John's ratings from previous borrowers.
- Jane agrees to the terms, and a smart contract is created, locking the agreed-upon fees in escrow.
- She borrows the drill for the weekend.
- Once the borrowing period ends, and Jane returns the drill, the smart contract releases the locked funds back to John, concluding the transaction.

Outcomes

- Benefits: Users like Jane access tools they need without purchasing them outright, while lenders like John earn fees by lending out underutilized tools.
- Trust and Security: The use of smart contracts ensures secure and automated transactions, reducing the risk of disputes or fraud.
- Community Building: Positive borrowing experiences and a reliable rating system foster a community of trust among users.

Challenges

- Scalability: Handling a growing number of transactions might pose scalability challenges for the blockchain network.
- User Adoption: Educating users about blockchain technology and encouraging their participation might be an initial hurdle.
- Regulatory Compliance: Ensuring compliance with local regulations regarding lending and liability might be complex in different jurisdictions.

Flow chart

Listing creation process

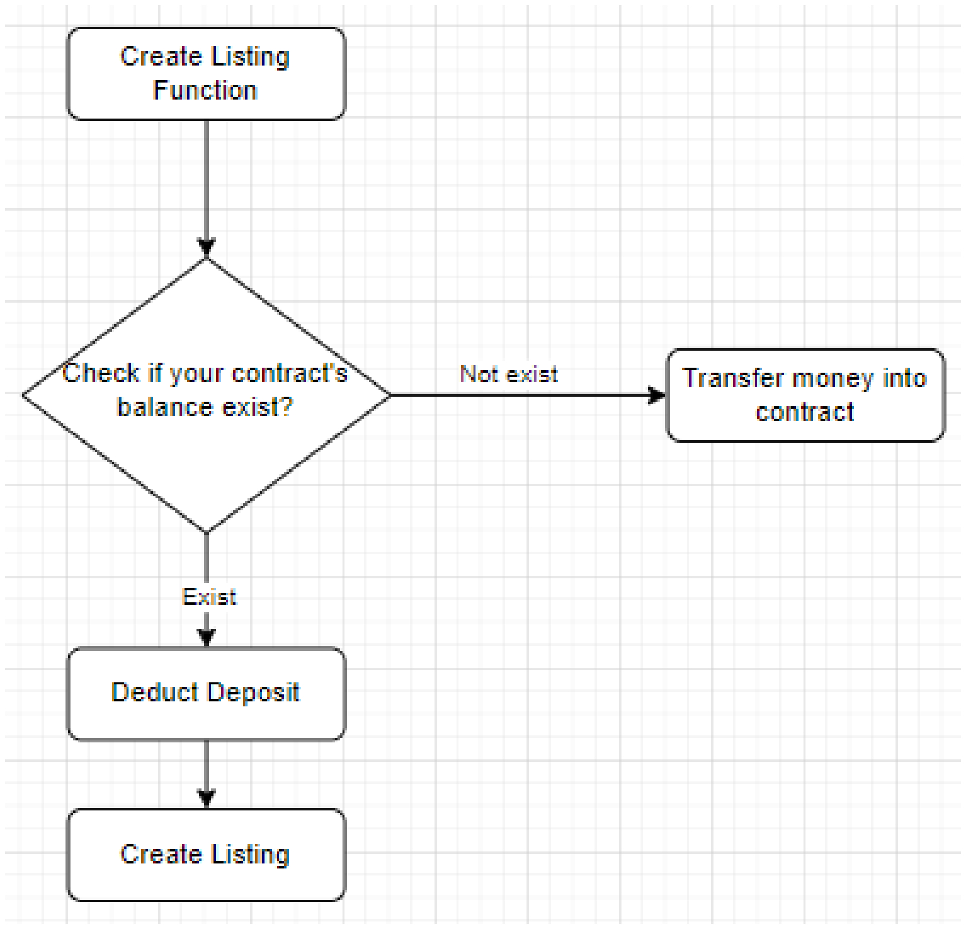

Create Listing Function
Check if your contract's balance exist?
Not exist
Transfer money into contract
Exist
Deduct Deposit
Create Listing

Figure 3.1 Flow chart of Listing creation process

Renting Process

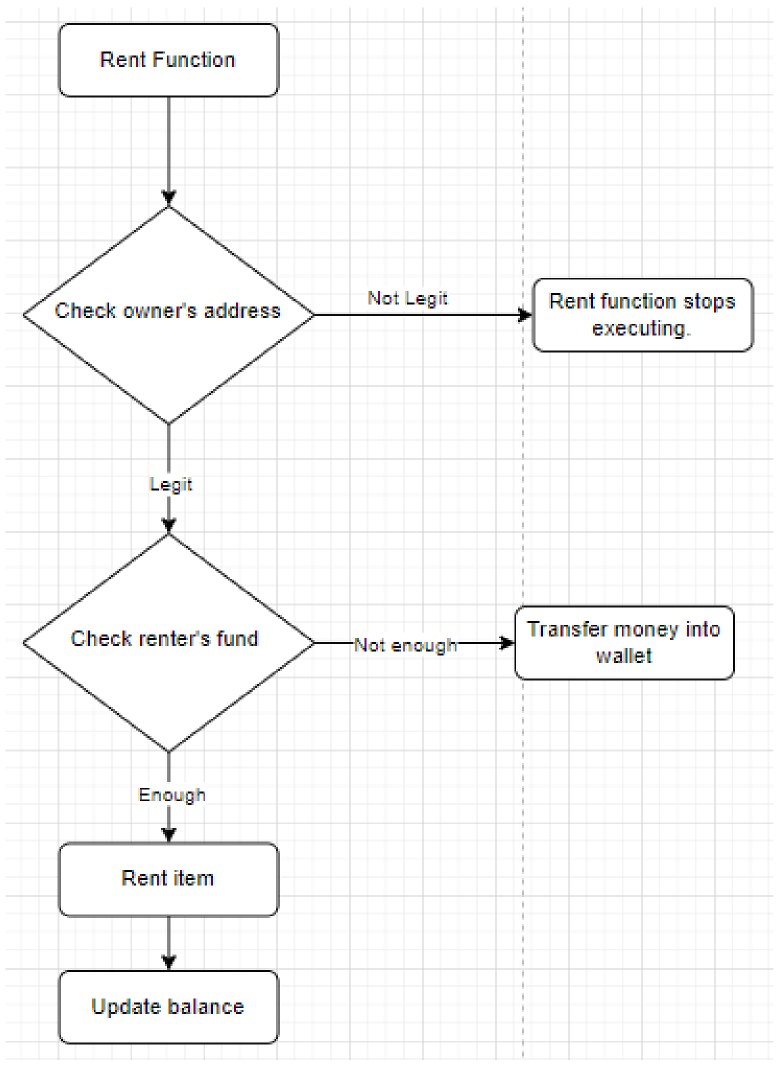


Figure 3.2 Flow chart of Renting Process

Returning Process

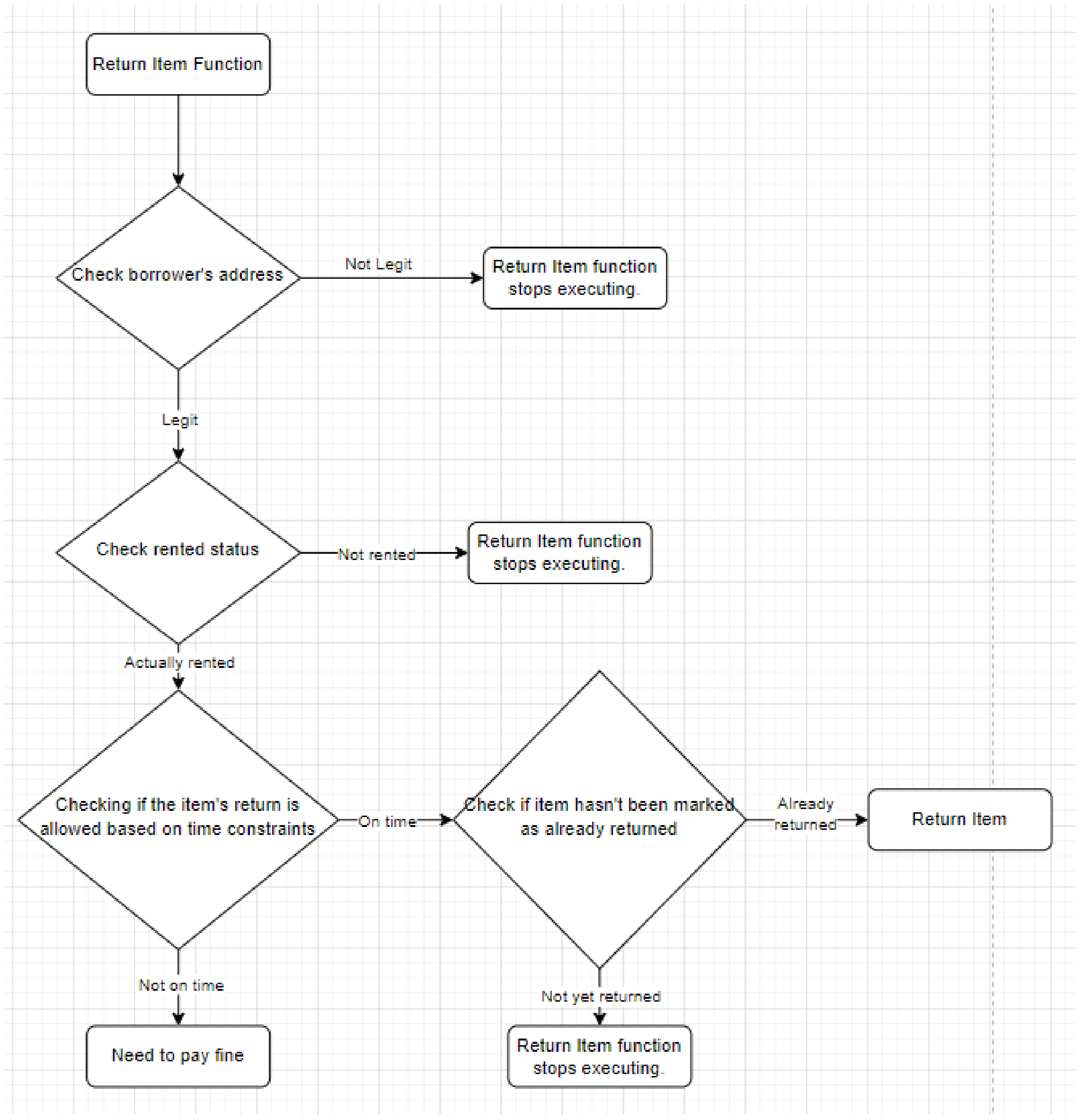

Return Item Function
Check borrower's address
Not Legit
Return Item function stops executing.
Legit
Check rented status
Not rented
Return Item function stops executing.
Actually rented
Checking if the item's return is allowed based on time constraints
On time
Check if item hasn't been marked as already returned
Already returned
Return Item
Not on time
Need to pay fine
Not yet returned
Return Item function stops executing.

Figure 3.3 Flow chart of Returning process

Deletion Process

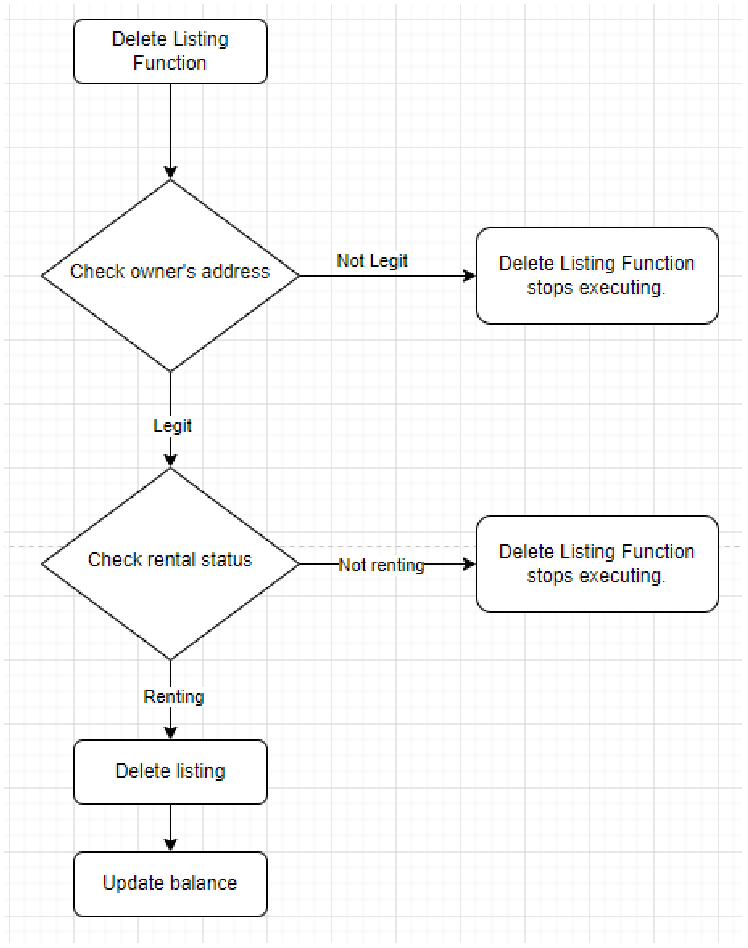


Figure 3.4 Flow chart of Deletion Process

Reporting non-returned item

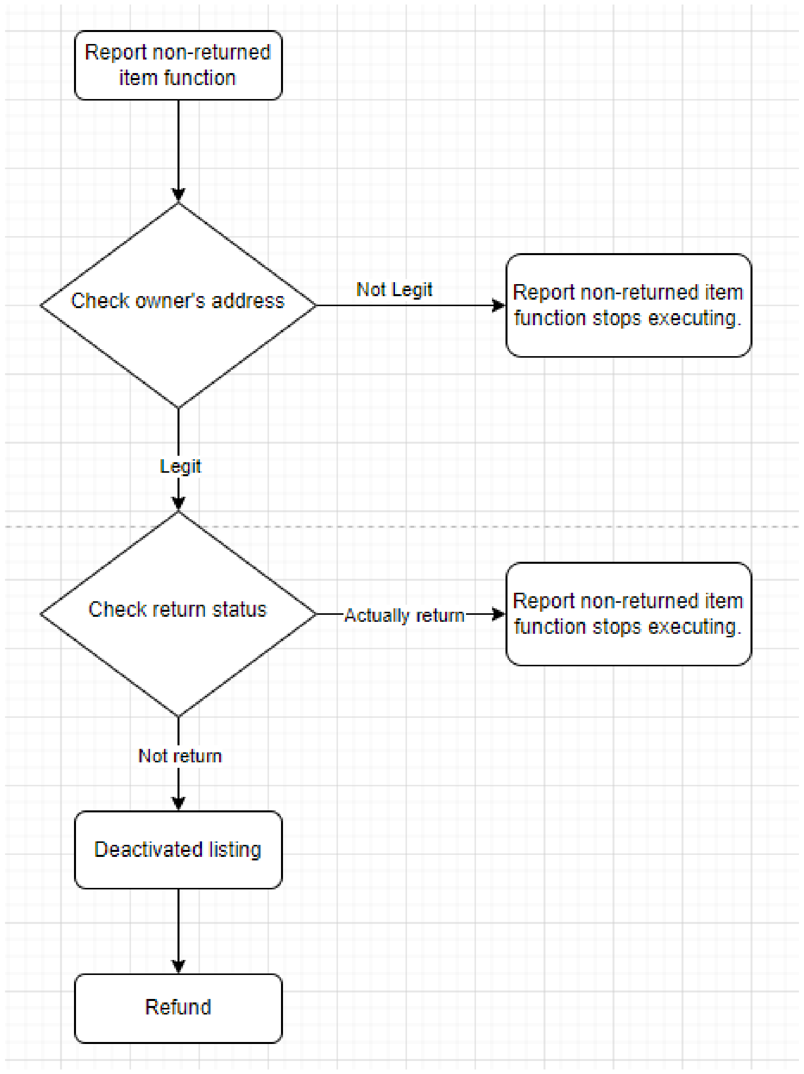

Report non-returned item function
Check owner's address
Not Legit
Report non-returned item function stops executing.
Legit
Check return status
Actually return
Report non-returned item function stops executing.
Not return
Deactivated listing
Refund

Figure 3.5 Flow chart of Reporting non-returned item

Framework and tools

Design

- Figma

  Figma is a cloud-based design and prototyping tool used for creating user interfaces, interactive prototypes, and collaborating on design projects in real-time. It's popular among UI/UX designers, product teams, and others involved in the creation of digital interfaces.

Version control

- Github

  GitHub is a platform for version control and collaborative software development. It allows developers to store and manage their code repositories in a centralized location using Git, a distributed version control system. Developers can work on projects individually or collaboratively, track changes made to the code, propose modifications, and merge those changes seamlessly.

Smart contract framework

- thirdweb

  thirdweb is a complete web3 development framework that provides everything you need to connect your apps and games to decentralized networks.

- Remix IDE

  used for the entire journey of smart contract development by users at every knowledge level. It requires no setup, fosters a fast development cycle, and has a rich set of plugins with intuitive GUIs. The IDE comes in two flavors (web app or desktop app) and as a VSCode extension.

Frontend framework

- Next.js

  Next.js is a React framework for building full-stack web applications. You use React Components to build user interfaces, and Next.js for additional features and optimizations. Next.js also abstracts and automatically configures tooling needed for React, like bundling, compiling, and more. This allows you to focus on building your application instead of spending time with configuration.

- thirdweb react SDK

  The React SDK uses React Query under the hood to expose a collection of query and mutation hooks, each with built-in caching, query invalidation, query retries, and more.

  Each hook (except for wallet/network management) wraps some functionality of the TypeScript SDK, which are made available as either a query hook to read data, or as a mutation hook to write transactions to the blockchain.

When mutations are called (when a user executes a transaction), query invalidation is automatically triggered to update the relevant queries that depend on the data that was changed. For example, when minting a new NFT, queries that view information about NFTs are re-fetched to load the new NFT automatically.

- MUI

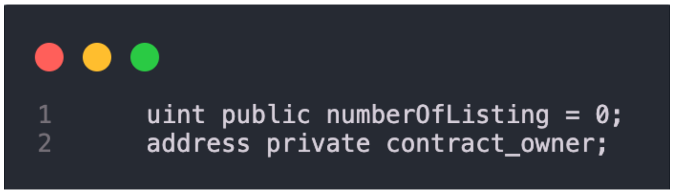
Material UI is an open-source React component library that implements Google's Material Design. It includes a comprehensive collection of prebuilt components that are ready for use in production right out of the box.

Material UI is beautiful by design and features a suite of customization options that make it easy to implement your own custom design system on top of our components.

Smart contract Development

Contract Structure

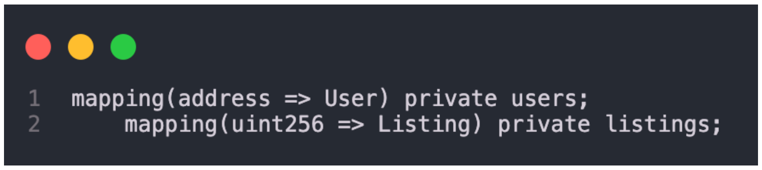
- State Variables
    - ‘numberOfListing’ : Tracks the total number of listings
    - ‘contract_owner’ : Stores the address of the contract creator



```
    uint public numberOfListing = 0;
    address private contract_owner;
```

Figure 4.1 State Variables

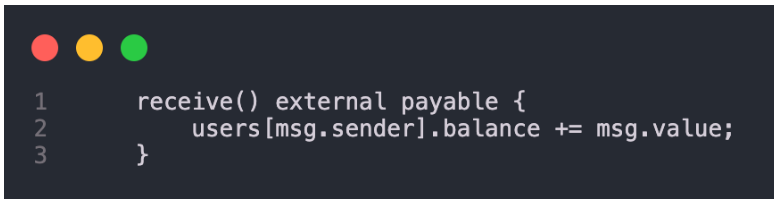
- Mappings
    - ‘user’ : Maps addresses to a ‘User’ struct
    - ‘listings’ : Maps listing IDs to a ‘Listing’ struct



```
mapping(address => User) private users;
    mapping(uint256 => Listing) private listings;
```

Figure 4.2 Mappings

- Receive Function
    - Allows the contract to receive Ether and adds it to the sender’s balance

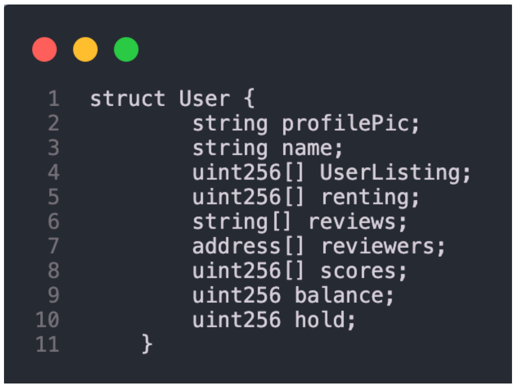


```
    receive() external payable {
        users[msg.sender].balance += msg.value;
    }
```

Figure 4.3 Receive Function

Structs

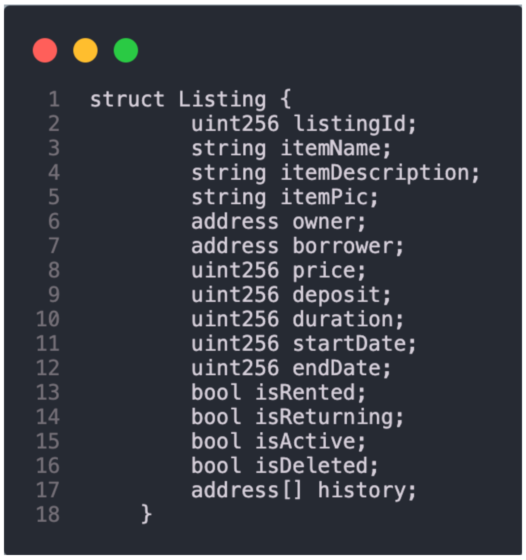
- User Struct
    - Stores user details, including listings,renting,history,reviews,scores and balances



```
struct User {
        string profilePic;
        string name;
        uint256[] UserListing;
        uint256[] renting;
        string[] reviews;
        address[] reviewers;
        uint256[] scores;
        uint256 balance;
        uint256 hold;
    }
```

Figure 4.4 User struct

- Listing Struct
    - Represents a rental item with various attributes such as ID, owner, borrower, price, deposit,duration, status etc.

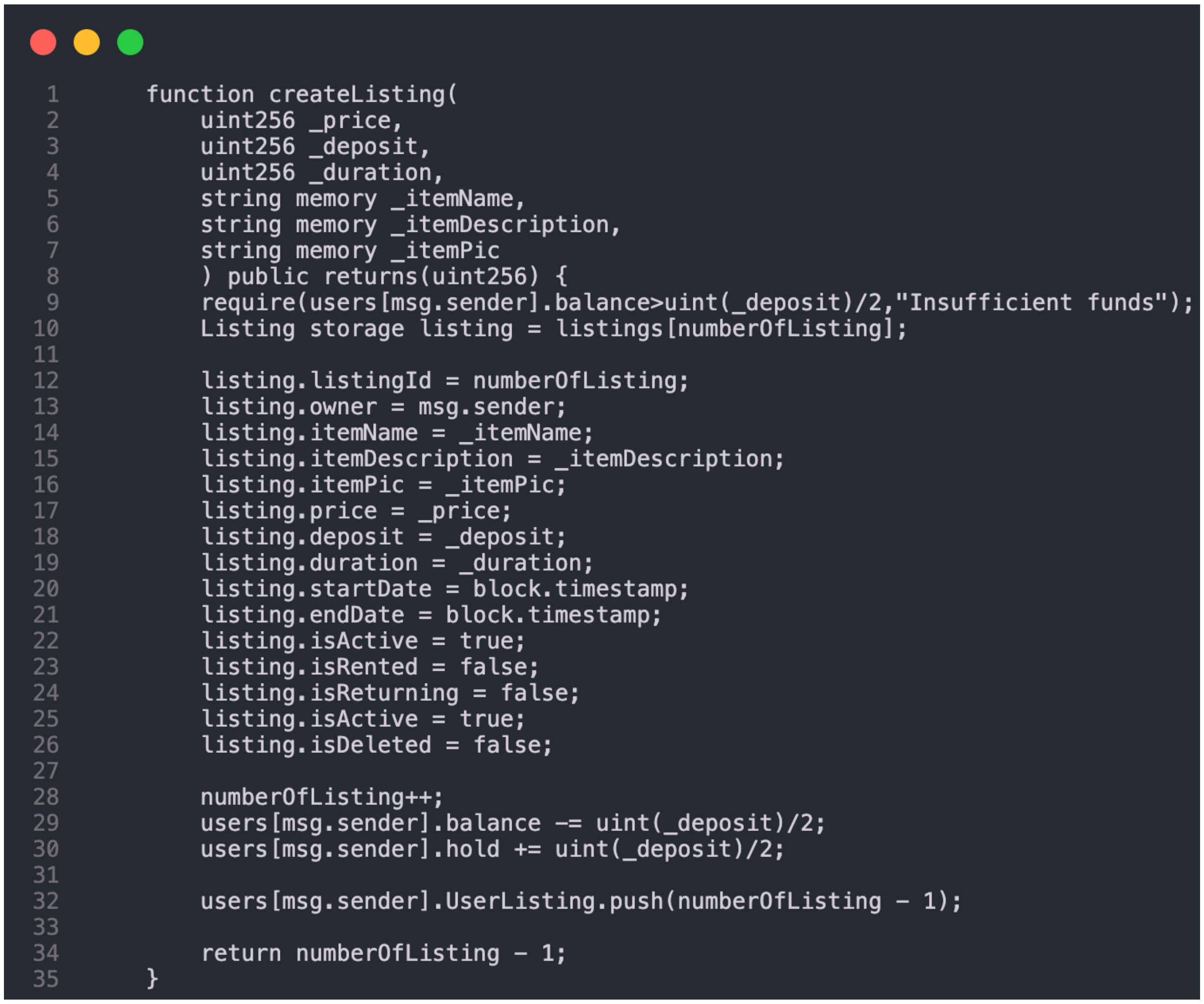


```
struct Listing {
        uint256 listingId;
        string itemName;
        string itemDescription;
        string itemPic;
        address owner;
        address borrower;
        uint256 price;
        uint256 deposit;
        uint256 duration;
        uint256 startDate;
        uint256 endDate;
        bool isRented;
        bool isReturning;
        bool isActive;
        bool isDeleted;
        address[] history;
    }
```

Figure 4.5 Listing struct

Functions

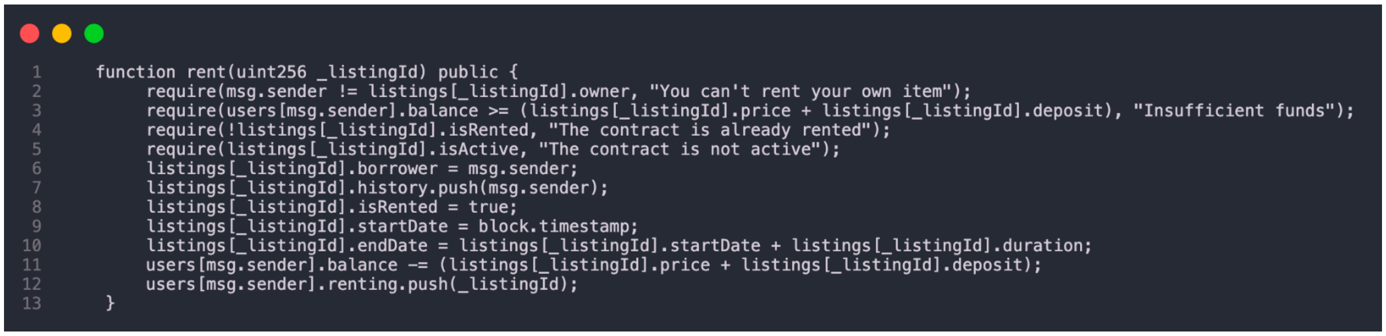
- createListing
    - Allows a user to create a new listing with specific details and deducts half of the deposit from the user's balance.



```
    function createListing(
        uint256 _price,
        uint256 _deposit,
        uint256 _duration,
        string memory _itemName,
        string memory _itemDescription,
        string memory _itemPic
        ) public returns(uint256) {
        require(users[msg.sender].balance>uint(_deposit)/2,"Insufficient funds");
        Listing storage listing = listings[numberOfListing];

        listing.listingId = numberOfListing;
        listing.owner = msg.sender;
        listing.itemName = _itemName;
        listing.itemDescription = _itemDescription;
        listing.itemPic = _itemPic;
        listing.price = _price;
        listing.deposit = _deposit;
        listing.duration = _duration;
        listing.startDate = block.timestamp;
        listing.endDate = block.timestamp;
        listing.isActive = true;
        listing.isRented = false;
        listing.isReturning = false;
        listing.isActive = true;
        listing.isDeleted = false;

        numberOfListing++;
        users[msg.sender].balance -= uint(_deposit)/2;
        users[msg.sender].hold += uint(_deposit)/2;

        users[msg.sender].UserListing.push(numberOfListing - 1);

        return numberOfListing - 1;
    }
```

Figure 4.6 createListing function

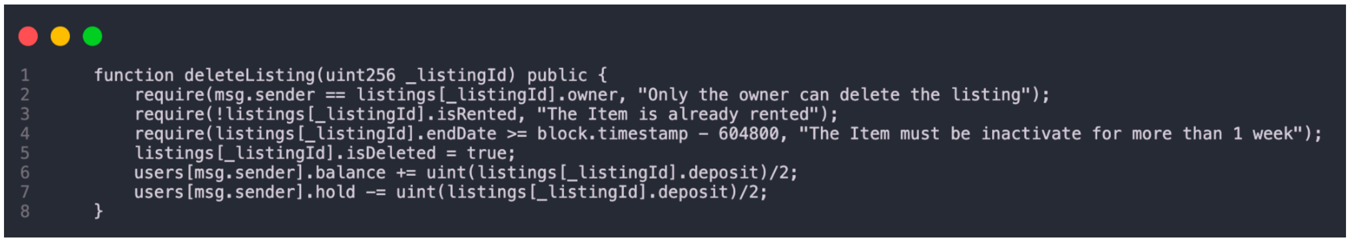
- rent
    - Enables users to rent items, deducting the necessary funds and updating item statuses.



```
function rent(uint256 _listingId) public {
     require(msg.sender != listings[_listingId].owner, "You can't rent your own item");
     require(users[msg.sender].balance >= (listings[_listingId].price + listings[_listingId].deposit), "Insufficient funds");
     require(!listings[_listingId].isRented, "The contract is already rented");
     require(listings[_listingId].isActive, "The contract is not active");
     listings[_listingId].borrower = msg.sender;
     listings[_listingId].history.push(msg.sender);
     listings[_listingId].isRented = true;
     listings[_listingId].startDate = block.timestamp;
     listings[_listingId].endDate = listings[_listingId].startDate + listings[_listingId].duration;
     users[msg.sender].balance -= (listings[_listingId].price + listings[_listingId].deposit);
     users[msg.sender].renting.push(_listingId);
 }
```

Figure 4.7 rent function

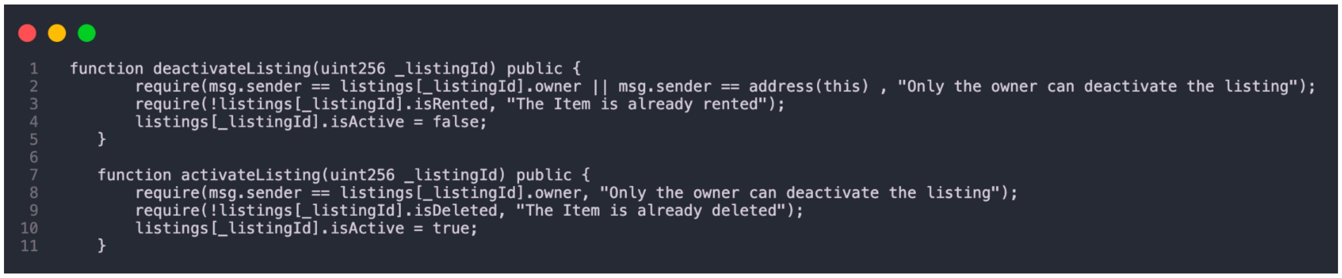
- deleteListing
    - Allows the owner to delete a listing if it's not rented and has been inactive for more than a week.



```
function deleteListing(uint256 _listingId) public {
    require(msg.sender == listings[_listingId].owner, "Only the owner can delete the listing");
    require(!listings[_listingId].isRented, "The Item is already rented");
    require(listings[_listingId].endDate >= block.timestamp - 604800, "The Item must be inactivate for more than 1 week");
    listings[_listingId].isDeleted = true;
    users[msg.sender].balance += uint(listings[_listingId].deposit)/2;
    users[msg.sender].hold -= uint(listings[_listingId].deposit)/2;
}
```

Figure 4.8 deleteListing function

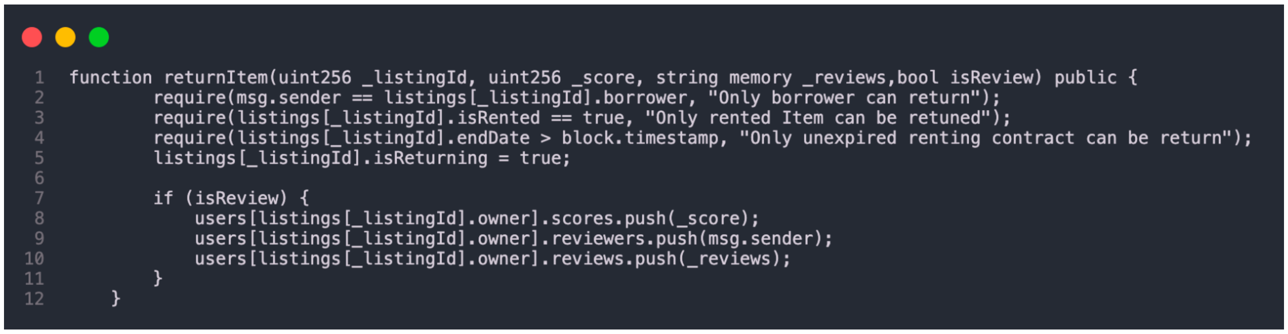
- deactivateListing/activateListing
    - Functions to deactivate/reactivate a listing.



```
function deactivateListing(uint256 _listingId) public {
        require(msg.sender == listings[_listingId].owner || msg.sender == address(this) , "Only the owner can deactivate the listing");
        require(!listings[_listingId].isRented, "The Item is already rented");
        listings[_listingId].isActive = false;
    }

    function activateListing(uint256 _listingId) public {
        require(msg.sender == listings[_listingId].owner, "Only the owner can deactivate the listing");
        require(!listings[_listingId].isDeleted, "The Item is already deleted");
        listings[_listingId].isActive = true;
    }
```

Figure 4.9 deactivateListing function and activateListing function

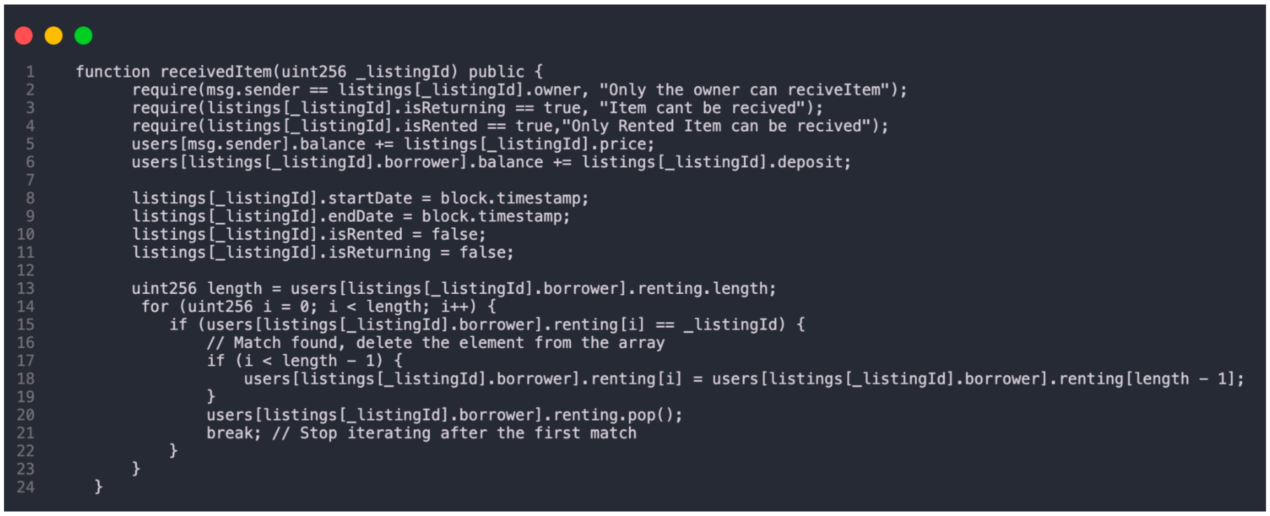
- returnItem:
    - Allows the borrower to return a rented item, potentially leaving a review for the owner.



```
function returnItem(uint256 _listingId, uint256 _score, string memory _reviews,bool isReview) public {
        require(msg.sender == listings[_listingId].borrower, "Only borrower can return");
        require(listings[_listingId].isRented == true, "Only rented Item can be retuned");
        require(listings[_listingId].endDate > block.timestamp, "Only unexpired renting contract can be return");
        listings[_listingId].isReturning = true;

        if (isReview) {
            users[listings[_listingId].owner].scores.push(_score);
            users[listings[_listingId].owner].reviewers.push(msg.sender);
            users[listings[_listingId].owner].reviews.push(_reviews);
        }
    }
```

Figure 4.10 returnItem function

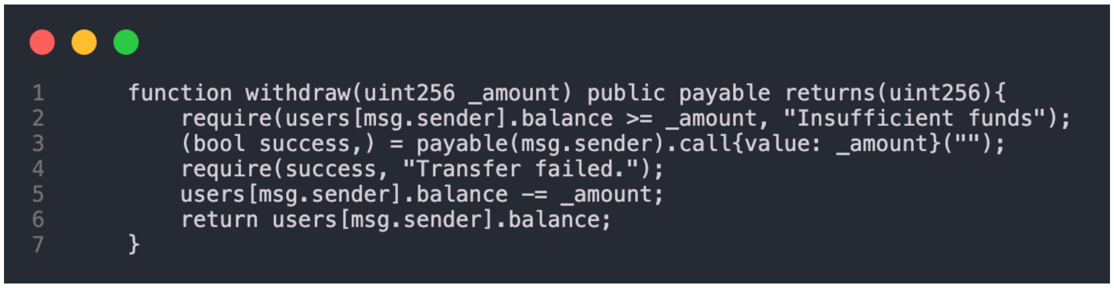
- receivedItem:
    - Marks an item as received by the owner after it's returned, updates balances, and removes the rental from the borrower's history.



```
function receivedItem(uint256 _listingId) public {
      require(msg.sender == listings[_listingId].owner, "Only the owner can reciveItem");
      require(listings[_listingId].isReturning == true, "Item cant be recived");
      require(listings[_listingId].isRented == true,"Only Rented Item can be recived");
      users[msg.sender].balance += listings[_listingId].price;
      users[listings[_listingId].borrower].balance += listings[_listingId].deposit;

      listings[_listingId].startDate = block.timestamp;
      listings[_listingId].endDate = block.timestamp;
      listings[_listingId].isRented = false;
      listings[_listingId].isReturning = false;

      uint256 length = users[listings[_listingId].borrower].renting.length;
       for (uint256 i = 0; i < length; i++) {
          if (users[listings[_listingId].borrower].renting[i] == _listingId) {
              // Match found, delete the element from the array
              if (i < length - 1) {
                  users[listings[_listingId].borrower].renting[i] = users[listings[_listingId].borrower].renting[length - 1];
              }
              users[listings[_listingId].borrower].renting.pop();
              break; // Stop iterating after the first match
          }
      }
  }
```

Figure 4.11 receivedItem function

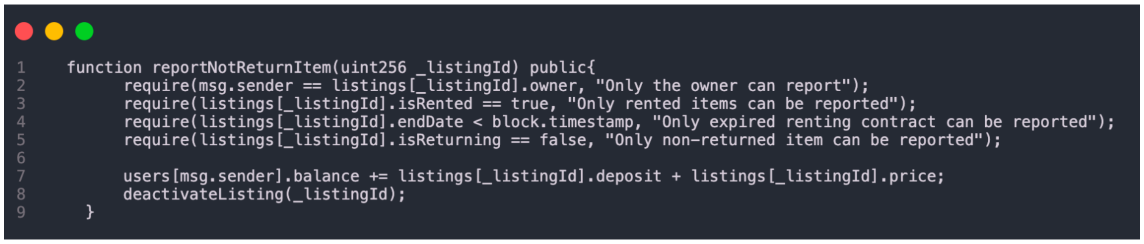
- withdraw:
    - Allows users to withdraw Ether from their contract balance.



```
function withdraw(uint256 _amount) public payable returns(uint256){
    require(users[msg.sender].balance >= _amount, "Insufficient funds");
    (bool success,) = payable(msg.sender).call{value: _amount}("");
    require(success, "Transfer failed.");
    users[msg.sender].balance -= _amount;
    return users[msg.sender].balance;
}
```

Figure 4.12 withdraw function

- reportNotReturnItem:
    - Enables the owner to report an item that hasn't been returned, deactivates the listing, and refunds the owner's balance.

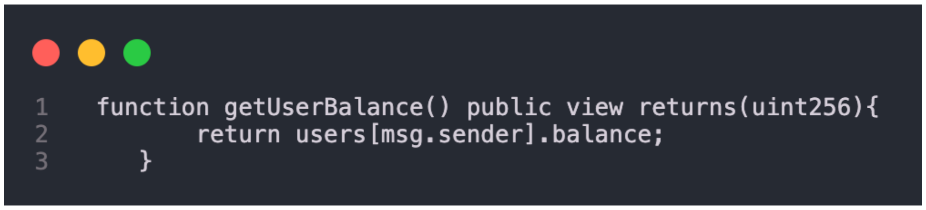


```
function reportNotReturnItem(uint256 _listingId) public{
      require(msg.sender == listings[_listingId].owner, "Only the owner can report");
      require(listings[_listingId].isRented == true, "Only rented items can be reported");
      require(listings[_listingId].endDate < block.timestamp, "Only expired renting contract can be reported");
      require(listings[_listingId].isReturning == false, "Only non-returned item can be reported");

      users[msg.sender].balance += listings[_listingId].deposit + listings[_listingId].price;
      deactivateListing(_listingId);
  }
```

Figure 4.13 reportNotReturnItem function

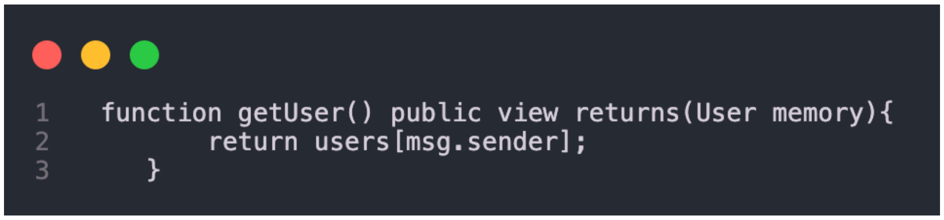
Getter Functions

- getUserBalance/getUser/getCurrentRenting/getUserListing/getUserByAddress/getActiveListings:
    - Provide various functionalities to retrieve user details, balances, user listings, active listings, etc.



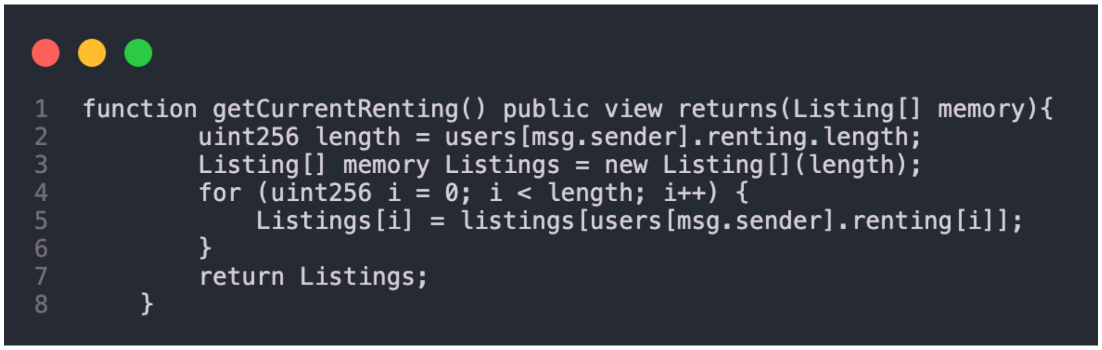
```
function getUserBalance() public view returns(uint256){
        return users[msg.sender].balance;
    }
```

Figure 4.14 getUserBalance function



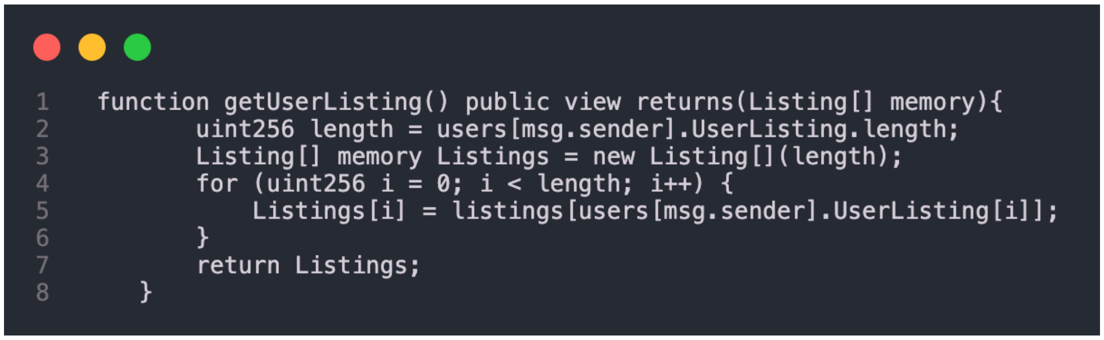
```
function getUser() public view returns(User memory){
        return users[msg.sender];
    }
```

Figure 4.15 getUser function



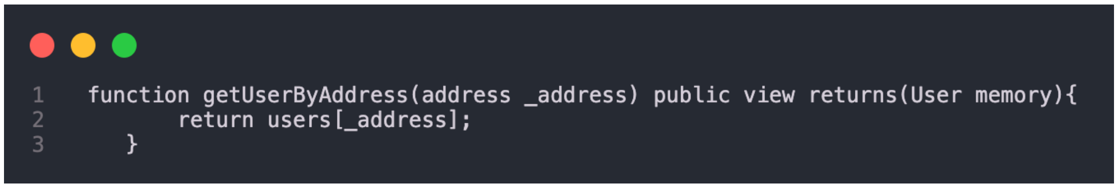
```
function getCurrentRenting() public view returns(Listing[] memory){
        uint256 length = users[msg.sender].renting.length;
        Listing[] memory Listings = new Listing[](length);
        for (uint256 i = 0; i < length; i++) {
            Listings[i] = listings[users[msg.sender].renting[i]];
        }
        return Listings;
    }
```

Figure 4.16 getCurrentRenting function



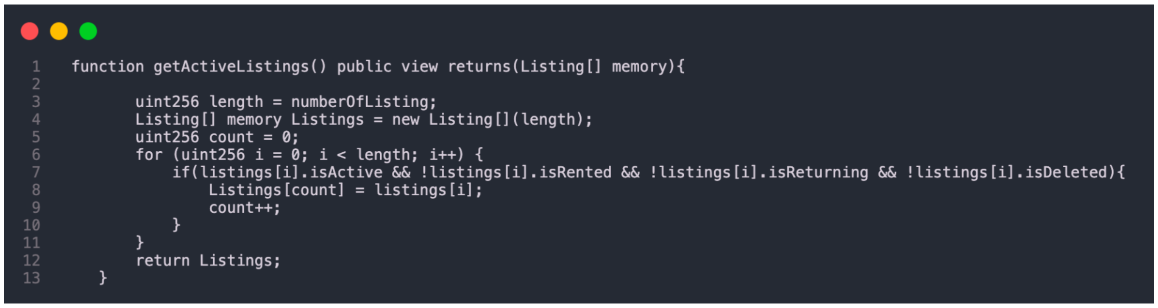
```
function getUserListing() public view returns(Listing[] memory){
        uint256 length = users[msg.sender].UserListing.length;
        Listing[] memory Listings = new Listing[](length);
        for (uint256 i = 0; i < length; i++) {
            Listings[i] = listings[users[msg.sender].UserListing[i]];
        }
        return Listings;
    }
```

Figure 4.17 getUserListing function



```
function getUserByAddress(address _address) public view returns(User memory){
        return users[_address];
    }
```

Figure 4.18 getUserByAddress function



```
function getActiveListings() public view returns(Listing[] memory){

        uint256 length = numberOfListing;
        Listing[] memory Listings = new Listing[](length);
        uint256 count = 0;
        for (uint256 i = 0; i < length; i++) {
            if(listings[i].isActive && !listings[i].isRented && !listings[i].isReturning && !listings[i].isDeleted){
                Listings[count] = listings[i];
                count++;
            }
        }
        return Listings;
    }
```

Figure 4.19 getActiveListings function

Fronted Development

Mockup

SHR

Figure 5.1 Logo of website

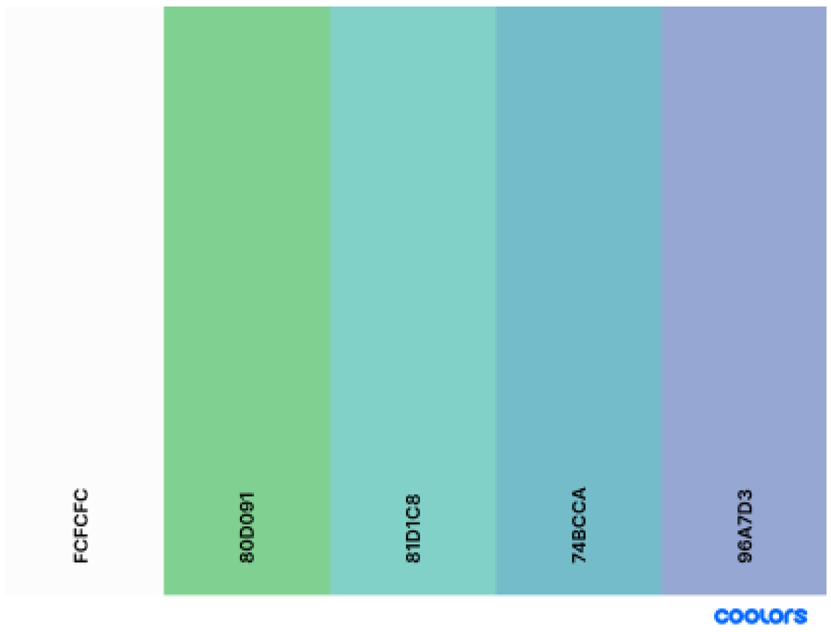


Figure 5.2 Color Palette

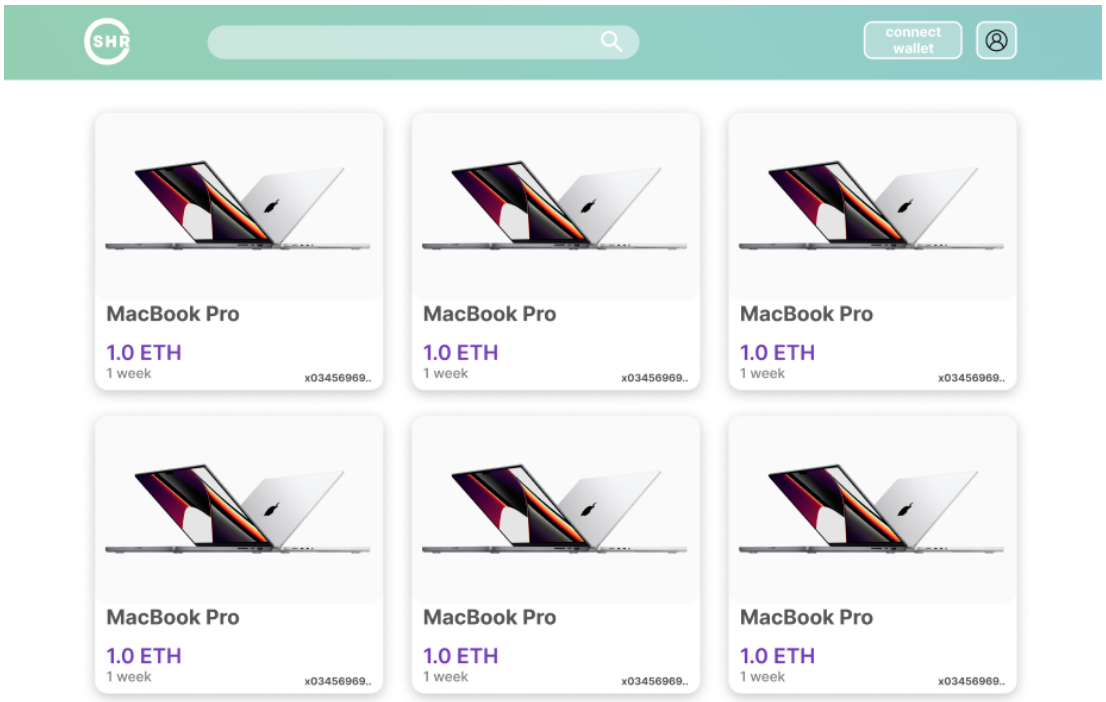


Figure 5.3 Home page

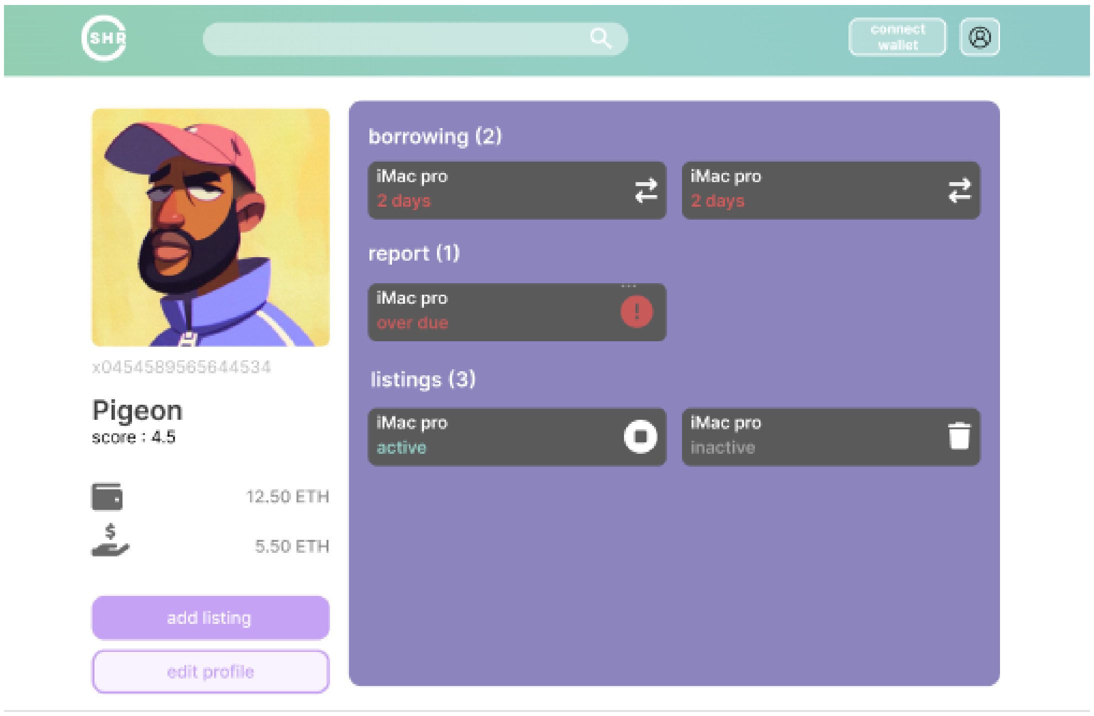
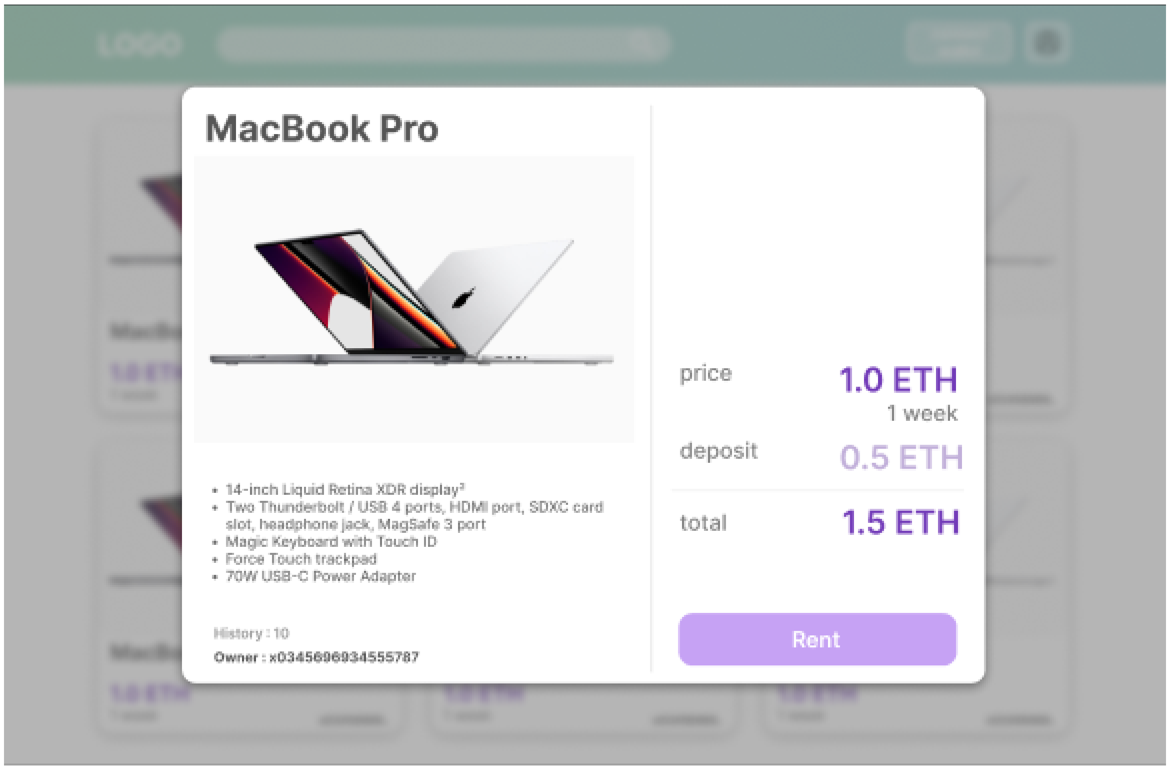


Figure 5.4 Profile page



MacBook Pro

- 14-inch Liquid Retina XDR display³
- Two Thunderbolt / USB 4 ports, HDMI port, SDXC card slot, headphone jack, MagSafe 3 port
- Magic Keyboard with Touch ID
- Force Touch trackpad
- 70W USB-C Power Adapter

History : 10

Owner : x0345696934555787

price	1.0 ETH 1 week
deposit	0.5 ETH
total	1.5 ETH

Figure 5.5 Detail modal

Further Improvements

- ### Voting System

When there's one situation that cannot make a decision, we will let investors decide by making a vote. We also provide rewards for investors who have already voted to ensure that they don't have fraud in the system.

- ### Collecting fee

We need to keep fees for maintaining the system and we also gives collected fees for community who makes a vote.

- ### Improving UX/UI

In the future, we will develop our UI to make users easier to use
For example, we will convert Wei into Eth form, we will change input from second(hard to read) to Minute, hour, day(More readable).

- ### Review Section

We also provide a review section to make the system more trustable so users can look for all histories to see if it's trustable or not.

Appendix

Github Link to the project

https://github.com/BlueFish01/web3-P2P-Rent