

**Sirindhorn International Institute of Technology**  
**CSS486 Blockchain Development – Semester II/2022 Capstone Project**

**Group Code: G09**

6222771824	Thiti Kungsapiwatana
6222771956	Lapbhavat Eaimsamang
6222782367	Gorrarith Bunnag

---

**Project Report: Storage System for Organizations**

**Introduction**

Blockchain is a technology that has the potential to revolutionize global industry and create a trust relationship in a multi-party business network, helps in the verification and traceability of multistep transactions needing verification. We also have to use the smart contract, that is programs that execute automatically when certain conditions or rules are met, making them particularly suitable for decentralized applications, since they can be used to process transactions, manage digital assets, store data, and automate processes in a distributed manner.

Our project goal is to create a decentralization application to manage physical equipment in an organization. We believe that with blockchain technology, it will provide transparency to manage and find information about the equipment in the stash. For backend development, we will use RemixIDE programming tool to write up smart contracts in solidity and for frontend development, we will use a local webpage to display the blockchain application using HTML, CSS and javascript.

## Case Study

We dugged down into one of the example use cases from ETHLend that use one of the functions of blockchain called peer-to-peer network system that provide individuals to lend and borrow funds directly from one another without the need for traditional financial intermediaries such as banks. Blockchain-based P2P lending platforms offer several advantages, including transparency, efficiency, and increased access to credit. ETHLend connects borrowers and lenders directly. Borrowers can create loan requests by specifying the loan amount, duration, and interest rate they are seeking. Lenders can browse through the loan requests and choose to fund the loans that align with their preferences.

ETHLend is a decentralized lending platform built on the Ethereum blockchain. It aims to provide individuals around the world with access to affordable loans without relying on banks or other financial institutions. We took a partial idea of this case study to implement it on our project. That is instead of borrowing a loan, we make it into borrowing an item instead.

## Requirements

Since our project aims to help organizations to manage their physical equipment, our project requirements will be simple.

1. Be able to add equipment into the stash. This is the most important requirement of our project since our goal is to manage physical equipment, we need to have the ability to add and store the data.
2. Be able to borrow equipment as well as to return the equipment.
3. Be able to check the status of the equipment. In case that we want to know who is borrowing the equipment and what time that they borrowed.
4. Be able to view all the items stored in the stash.
5. Be able to edit the name of the equipment. In case the equipment has their name changed, or when we want to replace the equipment with new one.

## Implementation Overview

We use RemixIDE with Solidity to create smart contracts for this system. Then deploy the smart contract to Sepolia test network via metamask.

We choose Sepolia testing network as our blockchain network because it is free to use and has a high block rate, which is good for testing and debugging.


Metamask acts like a gateway that links our smart contract to the blockchain network. Also acting as a wallet itself.

For our HTML web server, we use lite-server to act as a server and connect to the smart contract to interact with blockchain based on the user command.


## Smart Contract

The following is the function that is the backbone of our system. However some of the conditions are implemented in the HTML server side such as checking if the item is exist or not.

addItem function is to add item ID and its name into the system. In the smart contract, it will automatically store stash ID, item ID, item name, borrow flag as false, borrower name as empty string, and block timestamp.

```
function addItem(uint _itemid,string memory _itemName) public{  infinite gas
    stashcounter++;
    stashes[_itemid] = Stash(stashcounter, _itemid, _itemName, false, "", block.timestamp);
    keys.push(_itemid);
}
```

borrowItem function allows the user to borrow the item using item ID and their own name. This function will set the borrower name to the item, set borrow time and set the borrow flag to true. However, if the borrow flag is true, it will throw a message instead.

```
1
2     function borrowItem(uint _itemid, string memory _borrowerName) public {  infinite gas
3         require(!stashes[_itemid].isBorrowed,"Item is being borrowed");
4
5         stashes[_itemid].borrowingBy = _borrowerName;
6         stashes[_itemid].borrowTime = block.timestamp;
7         stashes[_itemid].isBorrowed = true;
8     }
9
```

returnItem function is to return the item that the user has borrowed. It will require only the item ID. This function will set the borrow flag to false and set borrower name to empty string. There is also a require check to see if the borrow flag is false, it will throw a message.

```
0
1 function returnItem(uint _itemid) public {  infinite gas
2     require(stashes[_itemid].isBorrowed,"Item has not been borrowed");
3
4     stashes[_itemid].isBorrowed = false;
5     stashes[_itemid].borrowingBy = "";
6 }
```

changeItemName function is to change the name of existing item in stash.

```
26
27 function changeItemName(uint _itemid,string memory _itemName) public{  infinite gas
28     stashes[_itemid].itemName = _itemName;
29 }
30
```

gets function is to get the element from the block to be used in the other part (HTML server).

```
50
51 function getIsBorrowed(uint _itemid) public view returns (bool){  2895 gas
52
53     return stashes[_itemid].isBorrowed;
54 }
55
56
57 function getBorrowingBy(uint _itemid) public view returns (string memory){  infinite gas
58
59     return stashes[_itemid].borrowingBy;
60 }
61
62
63 function getItemName(uint _itemid) public view returns (string memory){  infinite gas
64     return stashes[_itemid].itemName;
65 }
66
67 function getBorrowTime(uint _itemid) public view returns (uint){  infinite gas
68     return stashes[_itemid].borrowTime;
69 }
70
```

```

33
34     function getKeys() public view returns (uint[] memory){  infinite gas
35         return keys;
36     }
37 }

```

sort function is for sorting the item id and storing it in an ascending order.

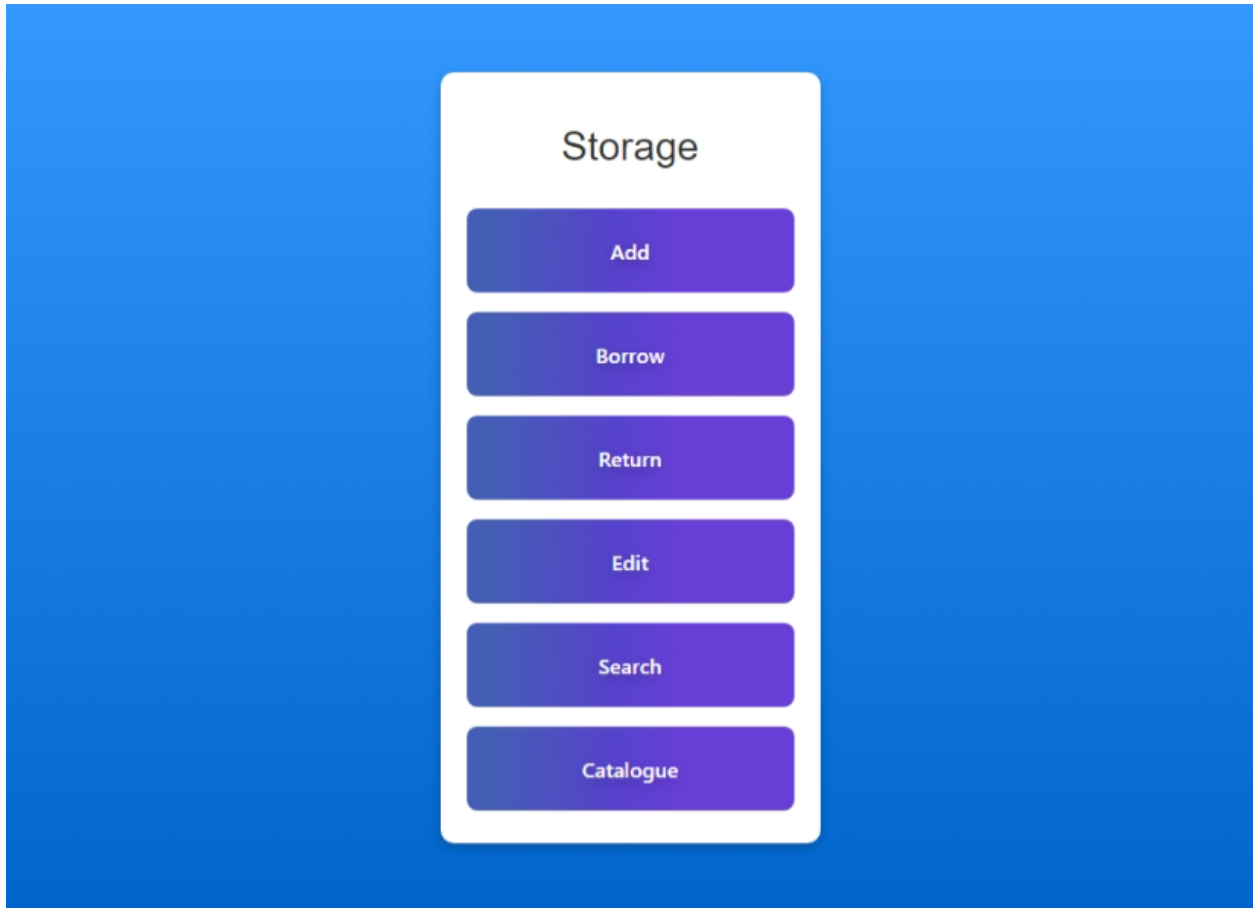
```

70
71     function sort() public {  infinite gas
72         uint length = keys.length;
73         for (uint i = 1; i < length; i++) {
74             uint key = keys[i];
75             int j = int(i) - 1;
76             while ((int(j) >= 0) && (keys[uint(j)] > key)) {
77                 keys[uint(j + 1)] = keys[uint(j)];
78                 j--;
79             }
80             keys[uint(j + 1)] = key;
81         }
82     }
83 }

```

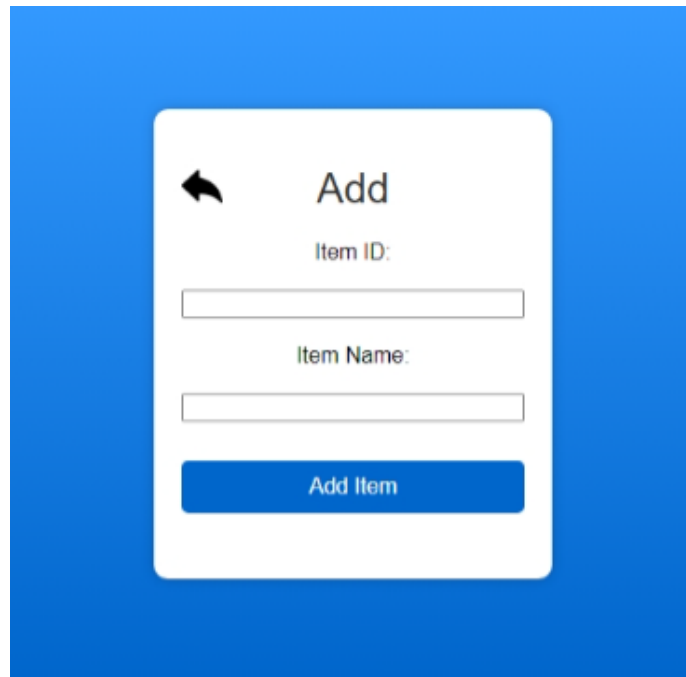
## Decentralized App

For the frontend of our dApp, we use HTML and CSS to make the simple UI.



In the main page of our application, we intended for simple usage, so we have 6 buttons that can be used to access the functions of the requirement of our application.

When we click the “Add” button from the home page, it will redirect to this page. Here, we can add items to stash using item id and item name. And we can go back to the main page by clicking on the arrow on top left.

A screenshot of a web form titled "Add" with a back arrow icon in the top left corner. The form is set against a blue gradient background. It contains two input fields: "Item ID:" and "Item Name:". Below these fields is a blue button labeled "Add Item".

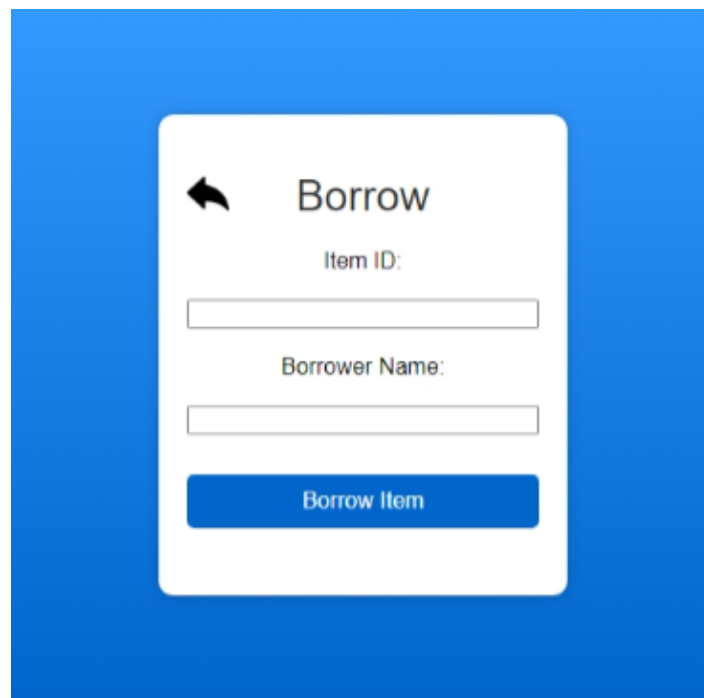
← Add

Item ID:

Item Name:

Add Item

In the borrow page, it is a place where we can borrow the equipment using item id and input the borrower name.

A screenshot of a web form titled "Borrow" with a back arrow icon in the top left corner. The form is set against a blue gradient background. It contains two input fields: "Item ID:" and "Borrower Name:". Below these fields is a blue button labeled "Borrow Item".

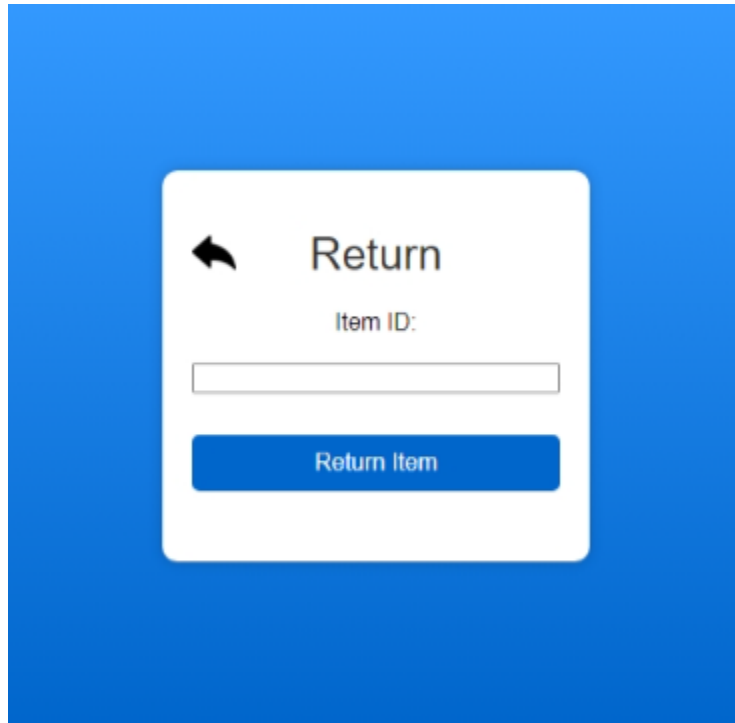
← Borrow

Item ID:

Borrower Name:

Borrow Item

When we borrow the equipment, we have to return it later as well. So in the return page, we put the item id of the equipment to return.

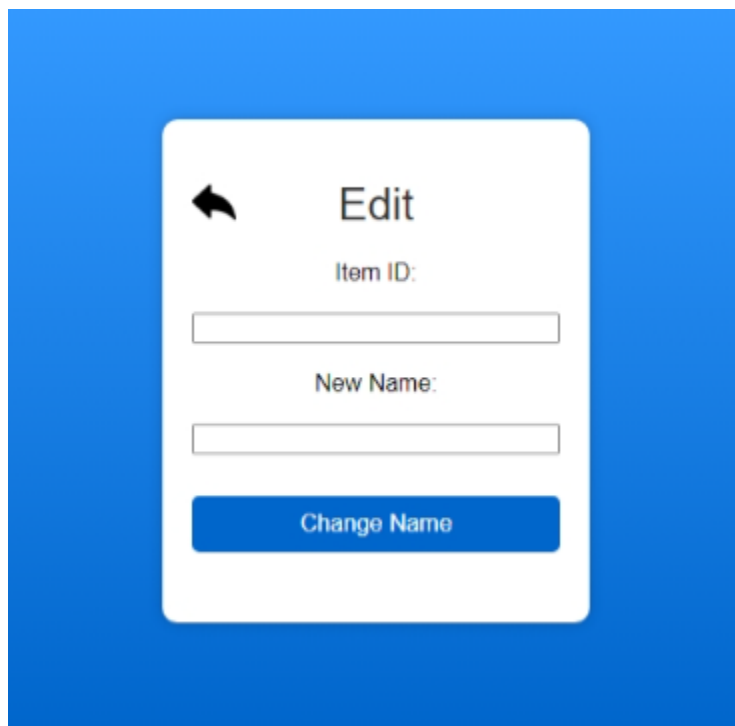
A screenshot of a web form titled "Return" with a back arrow icon. It features a label "Item ID:" above a text input field, and a blue button labeled "Return Item" at the bottom.

Return

Item ID:

Return Item

In the edit page, we can put the item id that we want to edit and put the new name.

A screenshot of a web form titled "Edit" with a back arrow icon. It features a label "Item ID:" above a text input field, a label "New Name:" above another text input field, and a blue button labeled "Change Name" at the bottom.

Edit

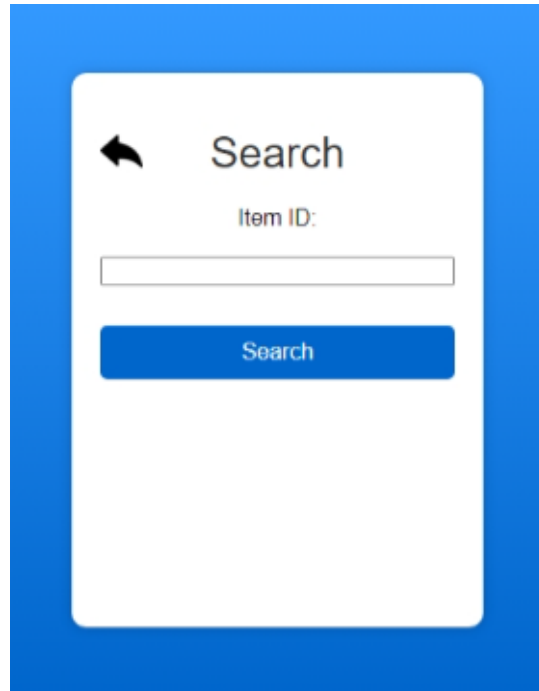
Item ID:

New Name:

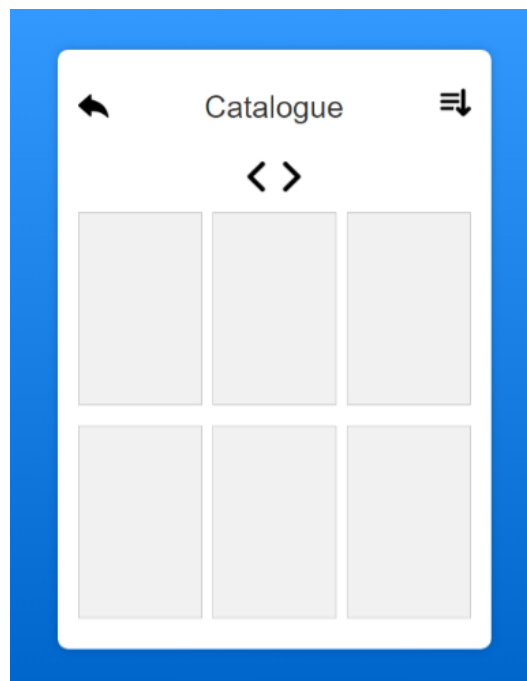
Change Name



In the search page, we can input the item id in the stash to check the status of that equipment. It will show item name, item status, borrower name (if the item is being borrowed) and time of borrowing (if the item is being borrowed).

The image shows a mobile app interface for a search page. It features a white rounded rectangle centered on a blue background. At the top left of the white area is a black back arrow icon. To its right is the word "Search" in a bold, black, sans-serif font. Below the title, the text "Item ID:" is displayed in a smaller, black font. Underneath this is a white rectangular input field with a thin grey border. At the bottom of the white area is a solid blue button with the word "Search" written in white, centered text.

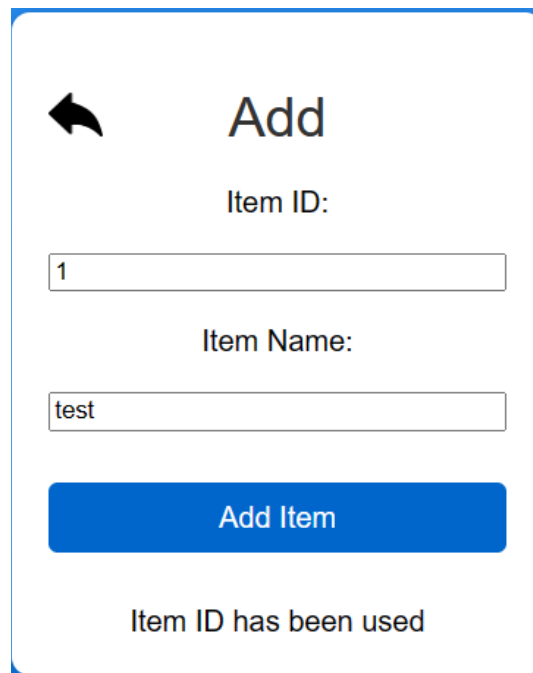
In the catalogue page, it shows 6 items in the stash at a time. We can click the arrow left/right in the top middle to move pages if there are more than 6 items. And we can sort items in ascending order by ID by clicking the icon on top right (Take some time to sort).

The image shows a mobile app interface for a catalogue page. It features a white rounded rectangle centered on a blue background. At the top left is a black back arrow icon. In the top center is the word "Catalogue" in a bold, black, sans-serif font. At the top right is a black sort icon consisting of three horizontal lines of decreasing length with a downward arrow. Below the title, there are two black chevron icons, one pointing left and one pointing right, separated by a small space. Below these are six grey rectangular placeholders arranged in a 2x3 grid, representing items in the catalogue.

## Test and Debug

We test and debug the system via the HTML web page using lite-server. We categorize the testing based on the function and following. Each function will also explain the expected output for each case.

**1. Add function:** This function allows you to add a new block to the chain using “Item ID” and “Item name” from the user. If the “Item ID” has been used already, it will show the feed back to the user.



The screenshot shows a web form titled "Add" with a back arrow icon. It contains two input fields: "Item ID:" with the value "1" and "Item Name:" with the value "test". Below the inputs is a blue "Add Item" button. At the bottom, a message states "Item ID has been used".

← Add

Item ID:

1

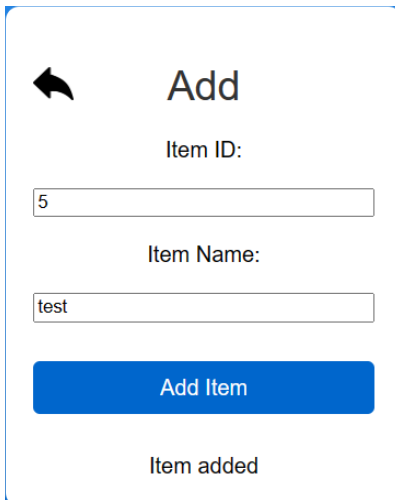
Item Name:

test

Add Item

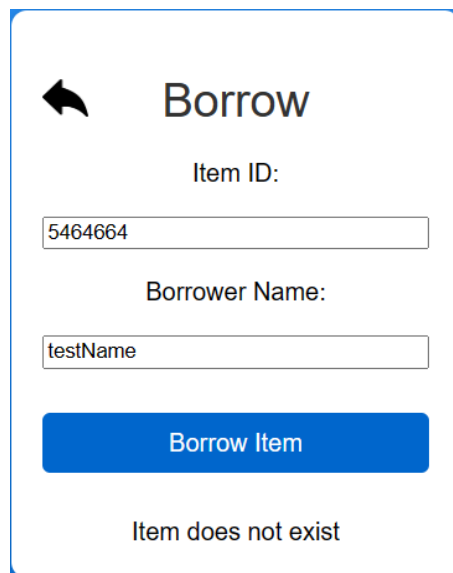
Item ID has been used

If the “Item ID” has not been used, it will add a new block to the chain with the corresponding “Item ID” and “Item Name” and show feedback to the user.




A mobile app interface for adding a new item. It features a back arrow icon in the top left corner. The title "Add" is centered at the top. Below the title, there are two input fields: "Item ID:" with the value "5" and "Item Name:" with the value "test". A blue button labeled "Add Item" is positioned below the input fields. At the bottom of the form, the text "Item added" is displayed.

**2. Borrow function:** This function is used to borrow items from the storage. Users must enter the "Item ID" of the item they are borrowing and the "Borrower Name". If the "Item ID" is not in the storage, it will show feedback to user as "Item does not exist"



A mobile app interface for borrowing an item. It features a back arrow icon in the top left corner. The title "Borrow" is centered at the top. Below the title, there are two input fields: "Item ID:" with the value "5464664" and "Borrower Name:" with the value "testName". A blue button labeled "Borrow Item" is positioned below the input fields. At the bottom of the form, the text "Item does not exist" is displayed.

If the Item is exist in the system but is being borrowed, it will show feedback to the user as follow



## Borrow


Item ID:

Borrower Name:

**Borrow Item**

name2.2 is being borrowed

Else it will allow users to borrow it.



## Borrow

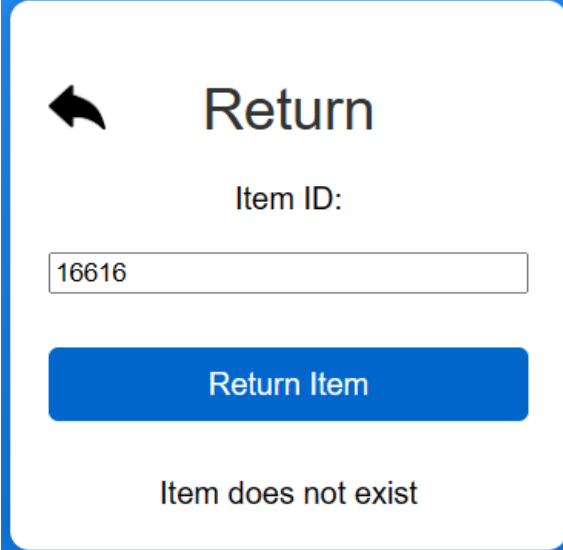
Item ID:

Borrower Name:

**Borrow Item**

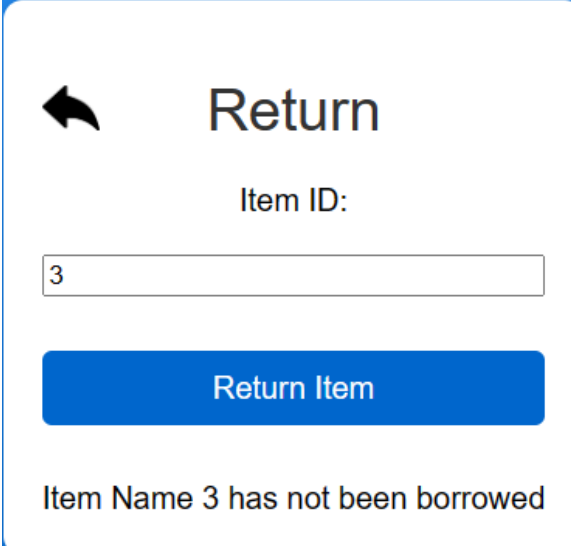
Borrow succeed

**3. Return function:** This function is used to return items to the storage. Users must enter the “Item ID” of the item they are returning. If the “Item ID” is not in the storage, it will show feedback to the user as “Item does not exist”.



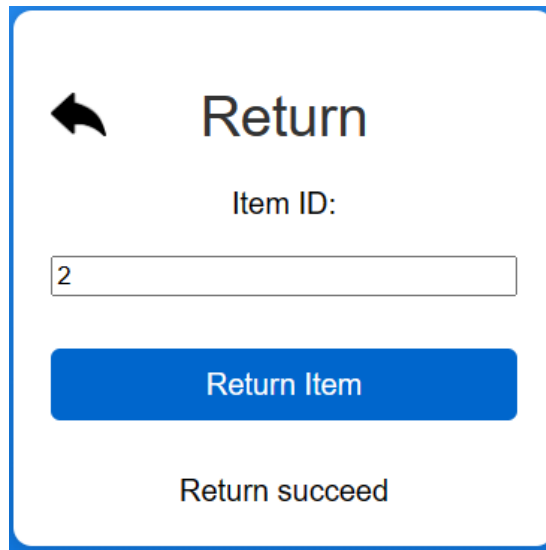
A screenshot of a web form titled "Return". It features a back arrow icon in the top left. Below the title is the label "Item ID:" followed by a text input field containing the value "16616". A blue button labeled "Return Item" is positioned below the input field. At the bottom of the form, the text "Item does not exist" is displayed.

If the Item exists in the system but the item has not been borrowed, it will show feedback to the user as follows.



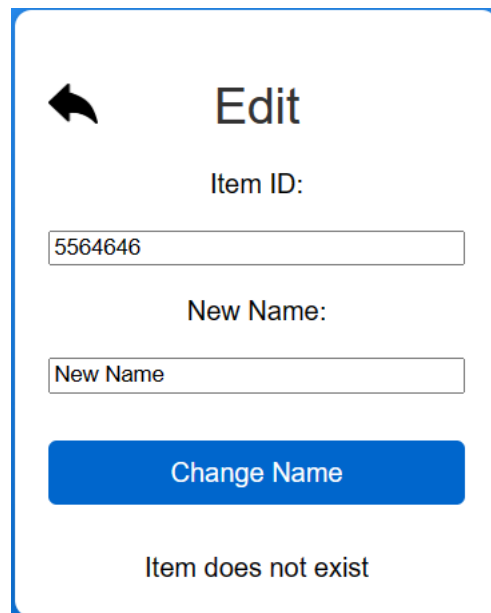
A screenshot of the same "Return" web form. The text input field now contains the value "3". The blue "Return Item" button remains below it. At the bottom of the form, the text "Item Name 3 has not been borrowed" is displayed.

Else it will let the user return the item.



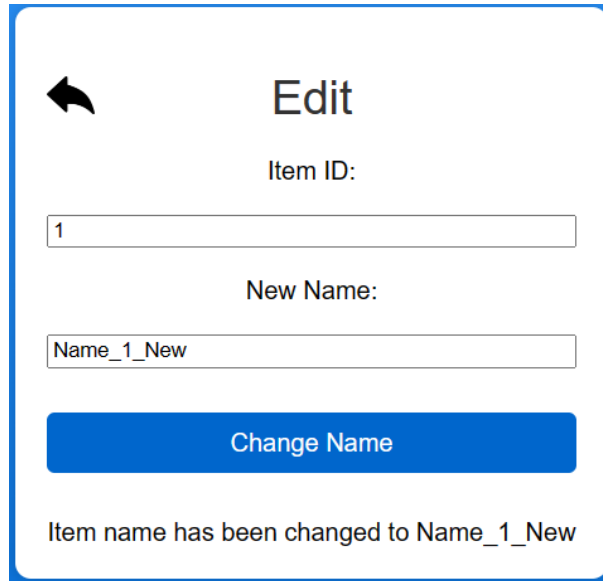
A mobile app interface for returning an item. It features a back arrow icon in the top left, a title 'Return' at the top center, and a label 'Item ID:' above a text input field containing the number '2'. Below the input field is a blue button labeled 'Return Item'. At the bottom, the text 'Return succeed' is displayed.

**4. Edit function:** This function allows the user to change the name of the item by entering “ItemID” and “New Name”. Then it will check if the item exists in the storage, if not it will show feedback to the user as follows.



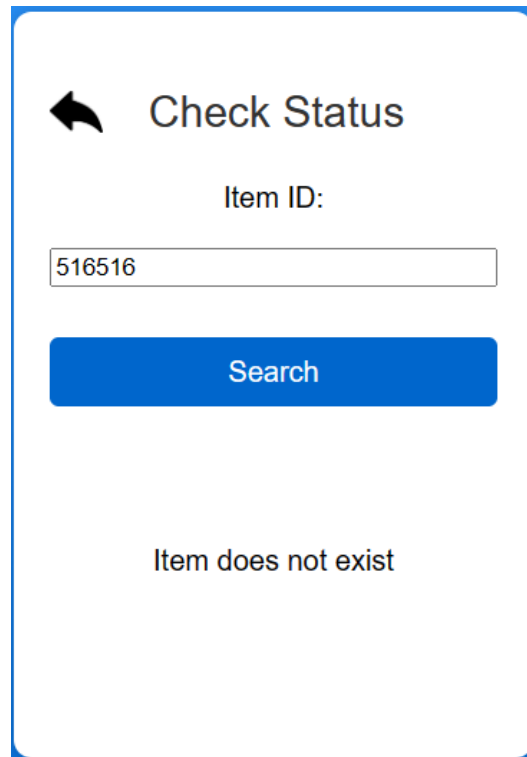
A mobile app interface for editing an item. It features a back arrow icon in the top left, a title 'Edit' at the top center, and a label 'Item ID:' above a text input field containing the number '5564646'. Below this is a label 'New Name:' above another text input field containing the text 'New Name'. A blue button labeled 'Change Name' is positioned below the second input field. At the bottom, the text 'Item does not exist' is displayed.

If the item exists, it will allow the user to change that item name and show the following feedback.



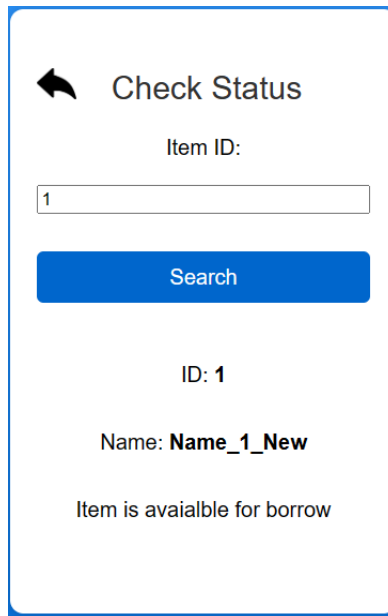
The image shows a mobile app interface for editing an item. It features a blue header bar with a back arrow icon on the left and the title "Edit" in the center. Below the header, there is a label "Item ID:" followed by a text input field containing the number "1". Underneath, there is a label "New Name:" followed by a text input field containing the text "Name\_1\_New". A blue button with the text "Change Name" is positioned below the input fields. At the bottom of the screen, a feedback message states "Item name has been changed to Name\_1\_New".

**5. Search function:** This function allows users to check the item status based on the "Item ID". If the item does not exist in the storage it will show the following feedback.



The image shows a mobile app interface for checking the status of an item. It features a blue header bar with a back arrow icon on the left and the title "Check Status" in the center. Below the header, there is a label "Item ID:" followed by a text input field containing the number "516516". A blue button with the text "Search" is positioned below the input field. At the bottom of the screen, a feedback message states "Item does not exist".

If the item exists in the storage and is available for borrow. It will show the item ID,name, and status as follows.



← Check Status

Item ID:

1

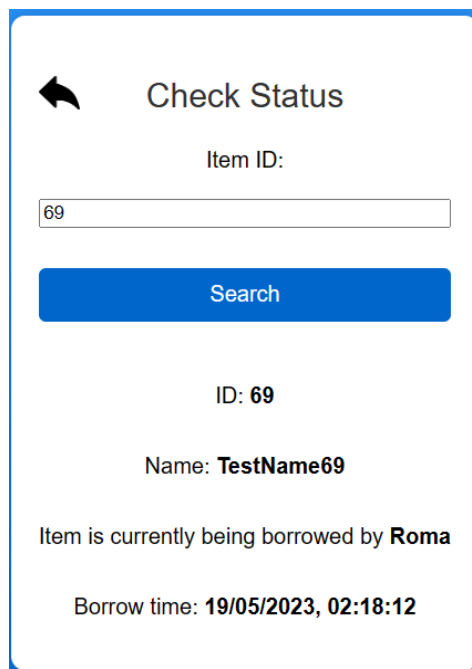
Search

ID: 1

Name: **Name\_1\_New**

Item is available for borrow

If the item exists in the storage and is currently being borrowed. It will show the item Id,item name, borrower name, and when it was borrowed as the follows.



← Check Status

Item ID:

69

Search

ID: 69

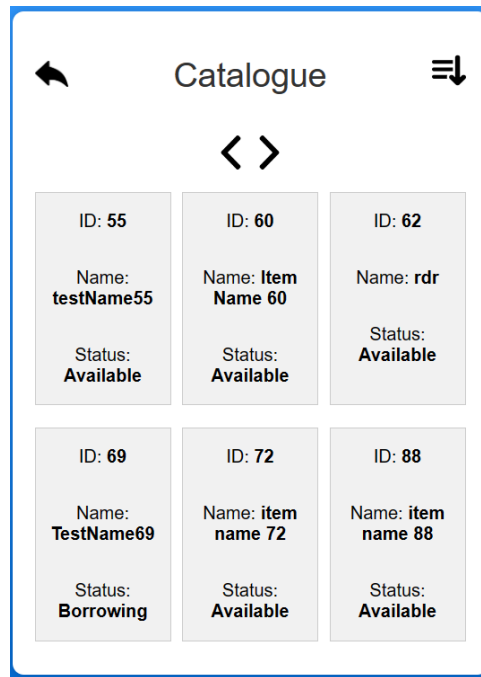
Name: **TestName69**

Item is currently being borrowed by **Roma**


Borrow time: **19/05/2023, 02:18:12**



**6. Get all items function:** This function shows 6 items at a time from the stash.



Users can change the page by clicking on the "<" or ">" icon.

Users can also sort the item in ascending order base on the item ID by clicking the  icon.

## Problem with Metamask

**Problem:** Metamask didn't recognise the borrow method in the transaction record part.

**Reason:** MetaMask uses a dictionary that translates function selector (0x and the first 8 hex characters of the data field) into a human-readable name.

If MetaMask doesn't find the function selector in the dictionary, they use a generic label "Contract Interaction".

To fix this issue, you need to manually add the method to the MetaMask dictionary.

## References

dApps tutorial

[Build your first dApp on Ethereum using HTML, Javascript, and Solidity! - DEV Community](#)

ETHLend White Paper

<https://github.com/ETHLend/Documentation/blob/master/ETHLendWhitePaper.md>