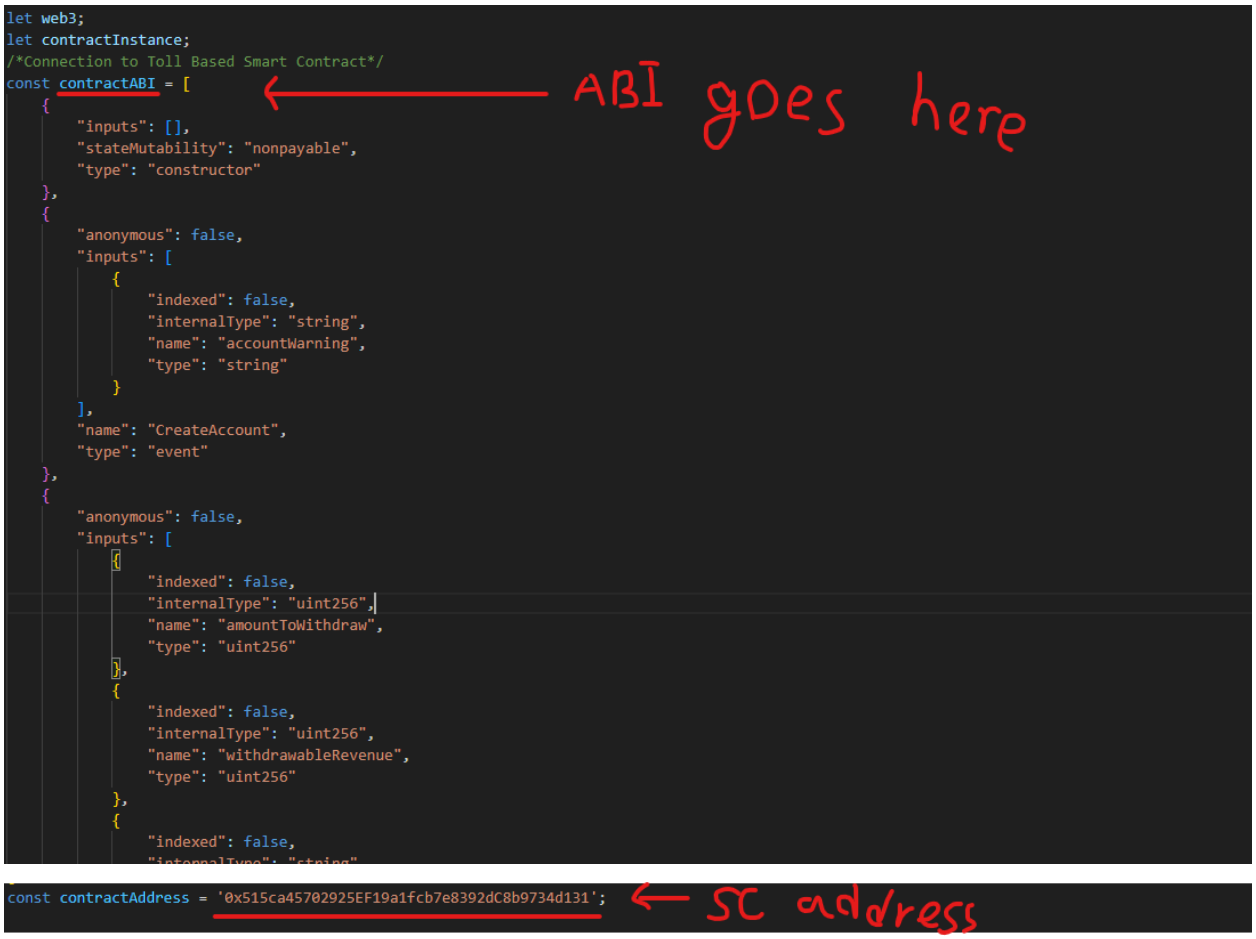
Developer Guide to working Front End of the Smart Contracts using javascript and node/web3 dependency

1. Update SC(Smart Contract) address and ABI(smart contract instance) in app.js



2. Set up a connection to users default browser cryptowallet

```
async function init() {
    /*Connect to the MetaMask Wallet*/
    if (window.ethereum) {
        console.log("Ethereum object found");  // Log when Ethereum object is found
        web3 = new Web3(window.ethereum);
        try {
            await window.ethereum.enable();
            console.log("Connection to MetaMask successful");  // Log when connection to MetaMask is successful
        } catch (error) {
            console.error("User denied account access");
        }
    } else if (window.web3) {
        console.log("Web3 object found");  // Log when web3 object is found
        web3 = new Web3(window.web3.currentProvider);
    } else {
        console.error("No web3 provider detected");
        return;
    }

    contractInstance = new web3.eth.Contract(contractABI, contractAddress);
    console.log("Contract instance created");  // Log when contract instance is created

    listenForEvents();
}
```

3. Add a Listener for the events, in this example, listener is waiting for the CreateAccount in Smart Contract

```
function listenForEvents() {
    /* Listener for each event in sc*/
    contractInstance.events.CreateAccount({}, (error, event) => {
        if (error) console.error(error);
        document.getElementById('CreateAccount').innerHTML = JSON.stringify(event);
    });

}
```

4. Here is an example of interaction with the function getAllIndUsersArrayItems from the tollbased smart contract

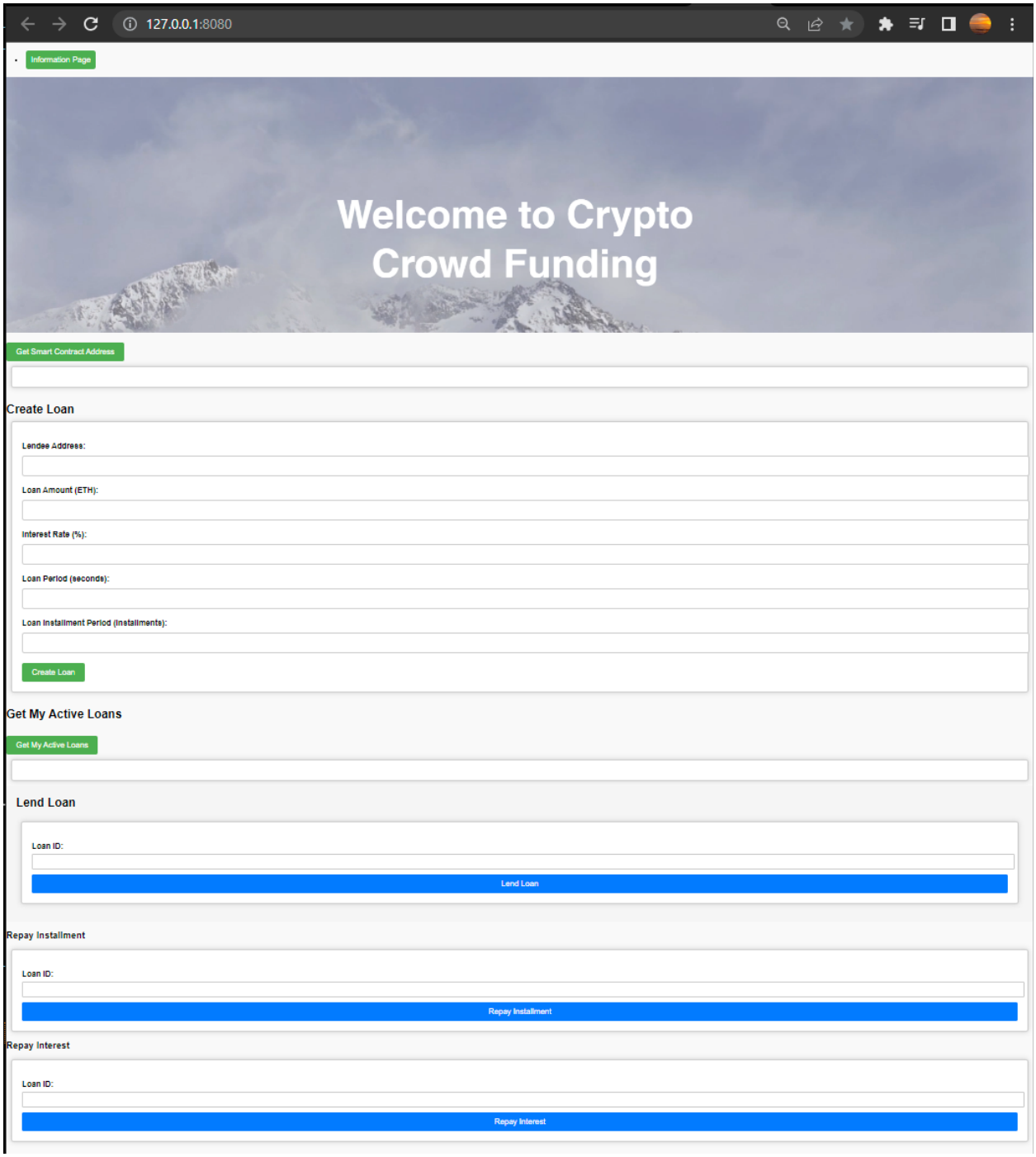
```
async function getAllIndUsersArrayItems() {
    try {
        const accounts = await web3.eth.getAccounts();
        await contractInstance.methods.getAllIndUsersArrayItems().send({from: accounts[0]});
        // Listen for the printIndUserArray event from the contract
        contractInstance.events.printIndUserArray((error, event) => {
            if (error) {
                console.error(error);
            } else {
                // Display the individual user array items
                console.log(event);
                const content = document.getElementById('getAllIndUsersArrayItemsResult');
                content.textContent += `First Name: ${event.returnValues[0]}, Last Name: ${event.returnValues[1]}, Wallet: ${event.returnValues[2]}, Total Payed: ${event.returnValues[3]}, User Type: ${event.returnValu
            }
        });
    } catch (error) {
        console.error(error);
    }
}
```

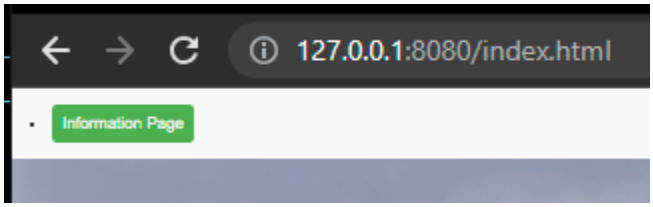
5. Lastly, everything we wrote in js file could be easily interacted with html files in order to display results and creating ui.

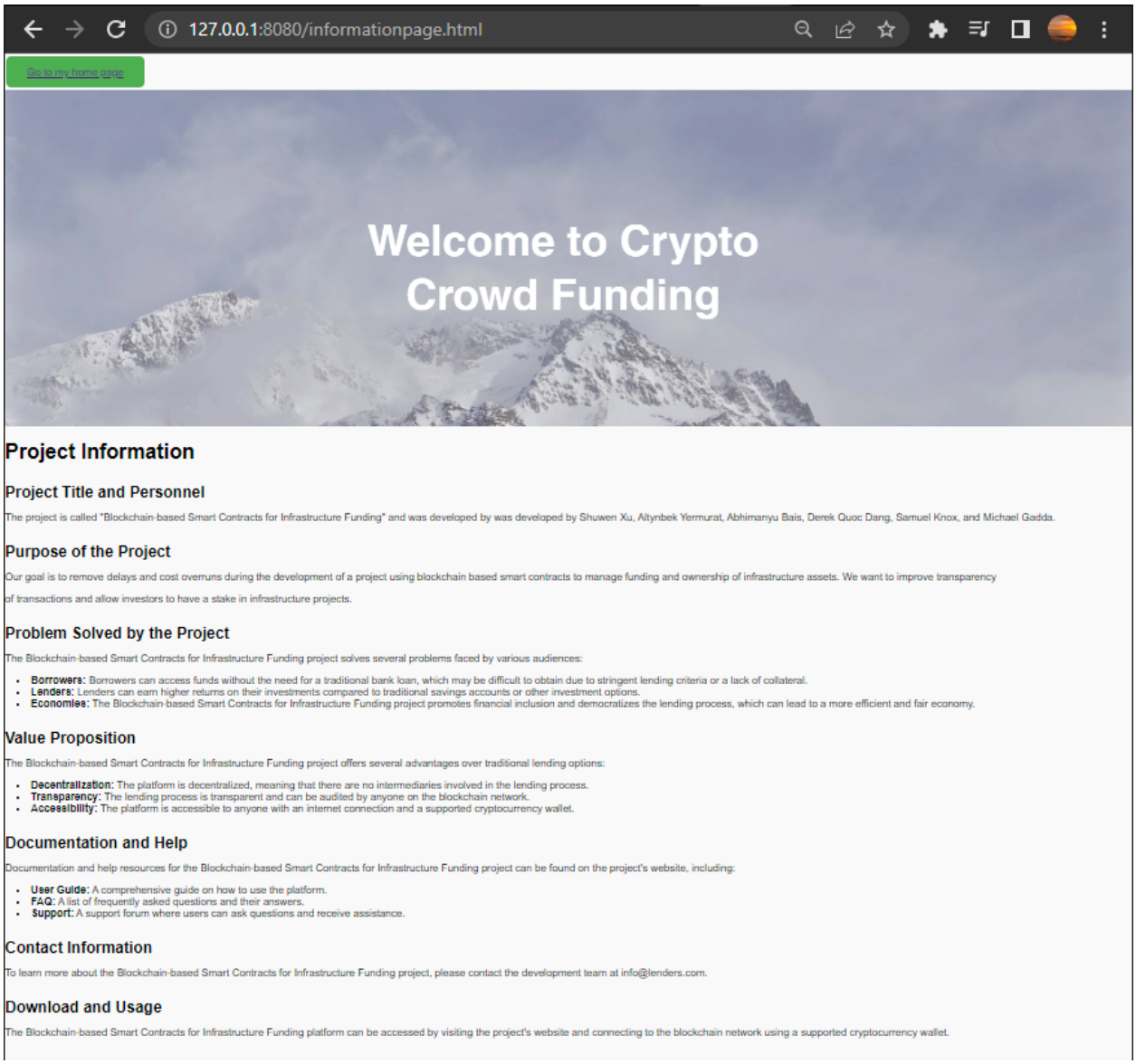
```
<h2>All Individual Users Array Items</h2>
<button onclick="getAllIndUsersArrayItems()">Get All</button>
<div id="getAllIndUsersArrayItemsResult"></div>
```

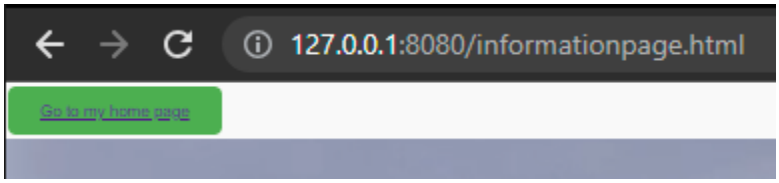
Basic interaction with the front-end guide:

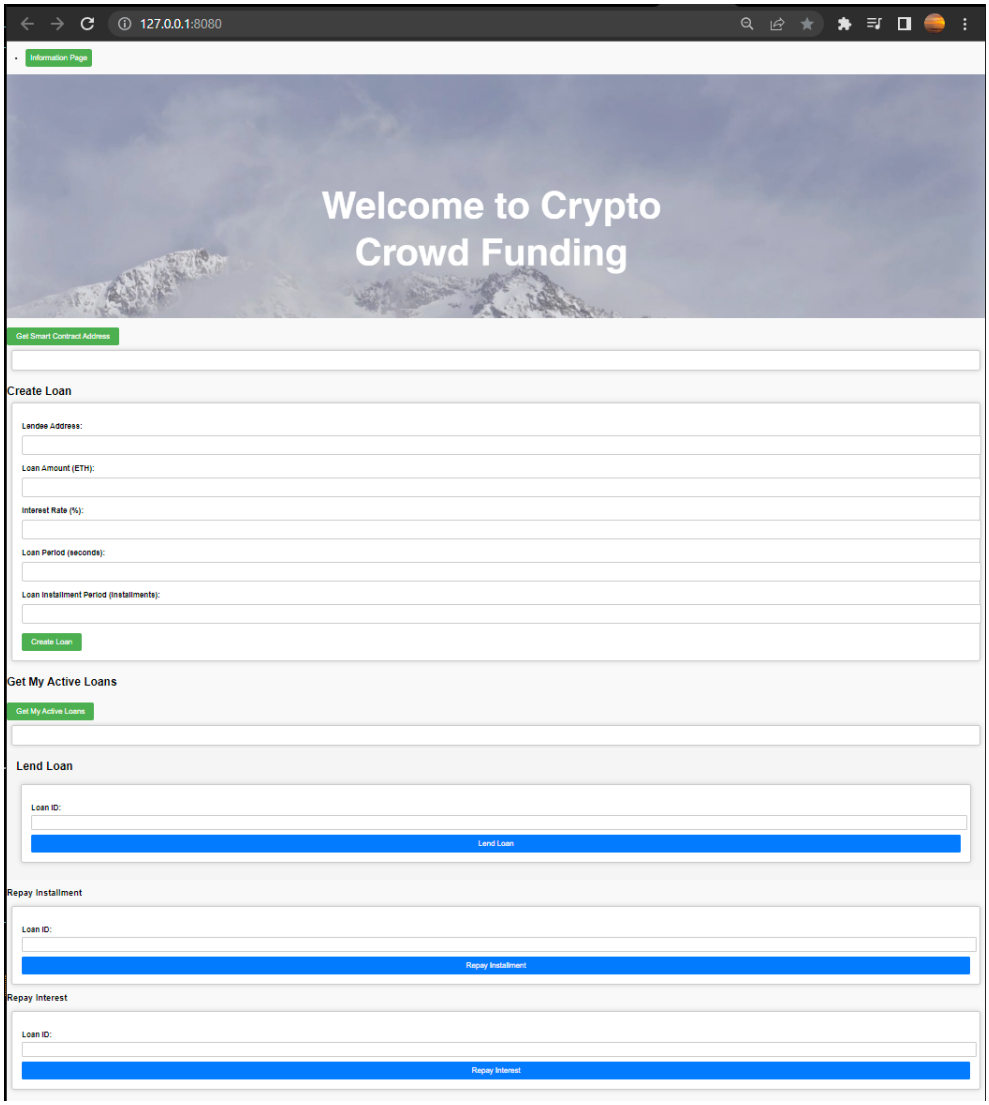
- **Starting out you will be greeted by this screen:**



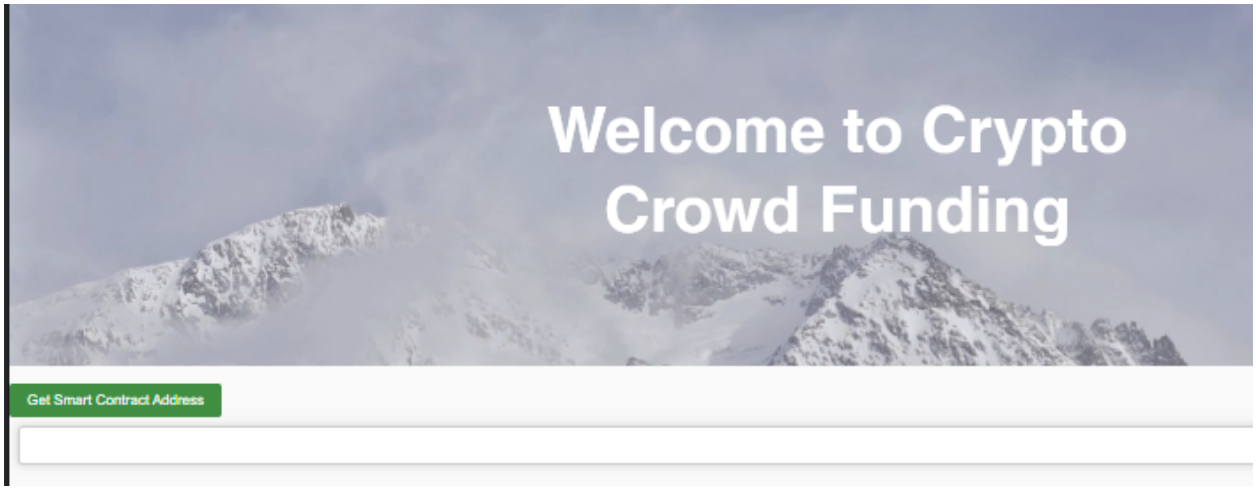
- **You can click on the information page button and go to a new page**
- 

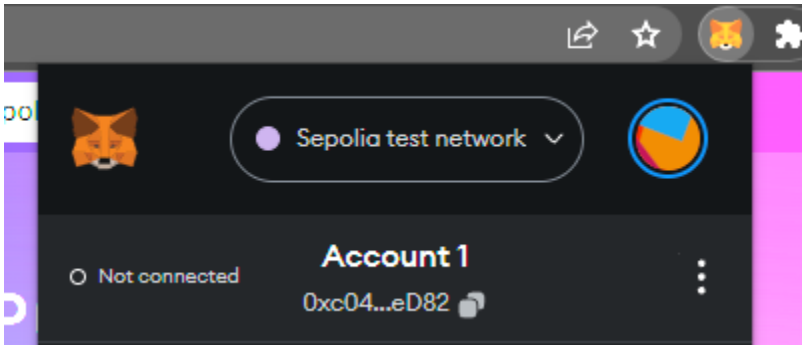
- **New information page this page has basic info on our capstone project**
- 

- You can click on the same button again to go back home
- 

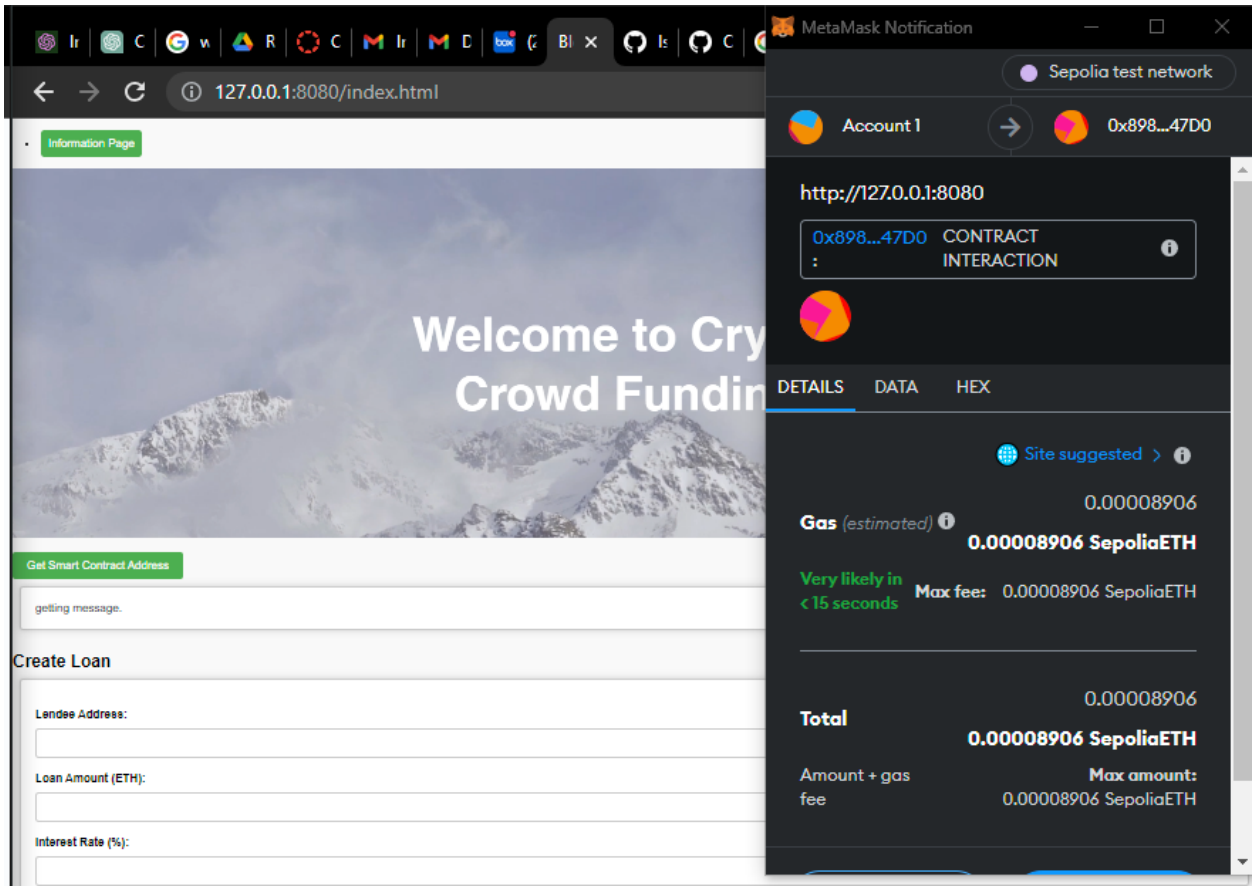
- And now your back home
- 

- The next button you can press and try out is the get smart contract address

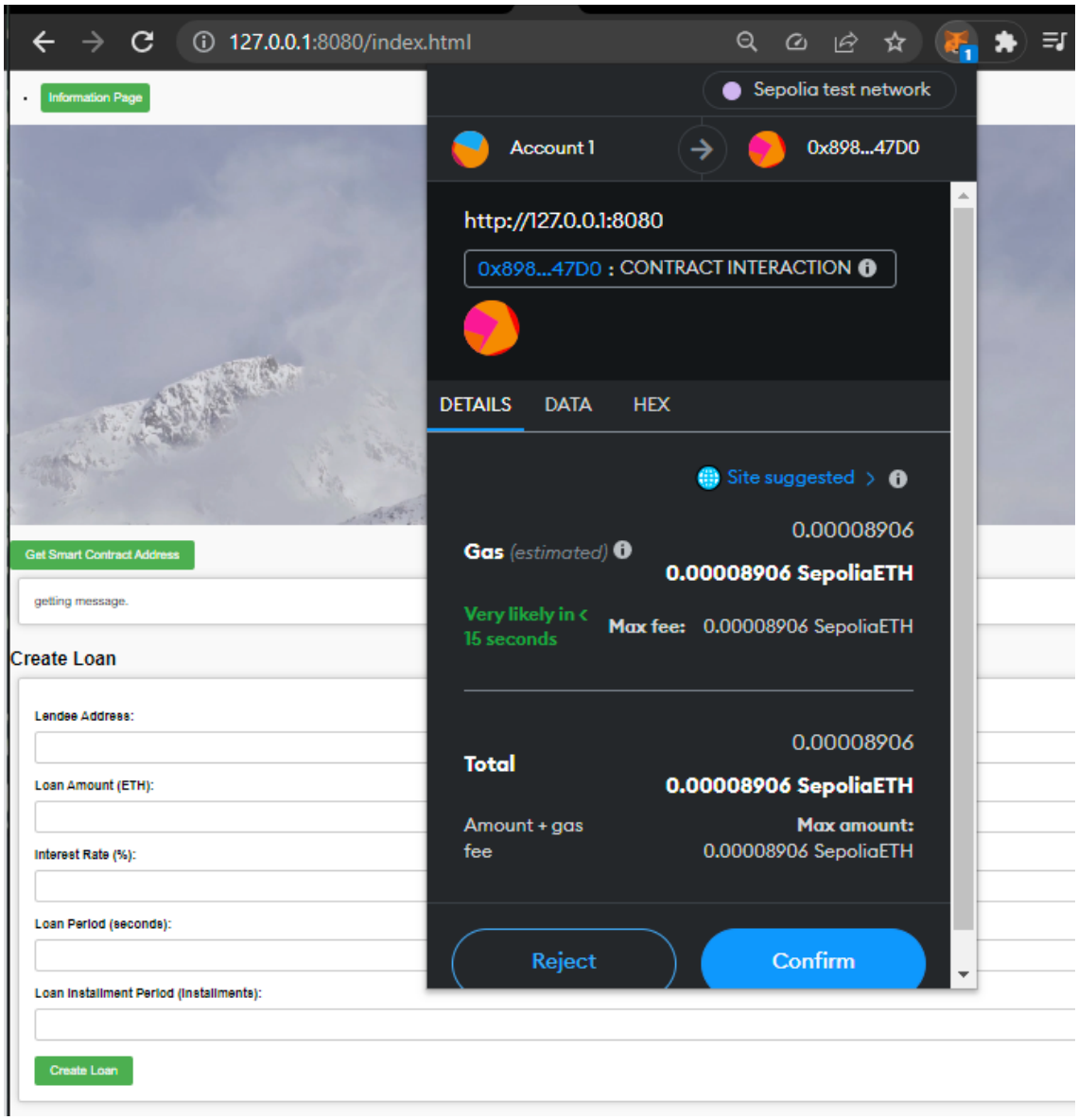
- 

- **There is an initial setup on meta mask where you have to set up an account I recommended for the sake of testing you ignore setting up a 12-word password setup for ur wallet but make sure that when it asks for the password you remember that**
- **Also, make sure you have some test eth (not actual Ethereum there are some fake coins for things like development) you can get some from a faucet: https://sepoliafaucet.com/ place ur wallet address which u can copy from the meta mask chrome plug in:**
- 

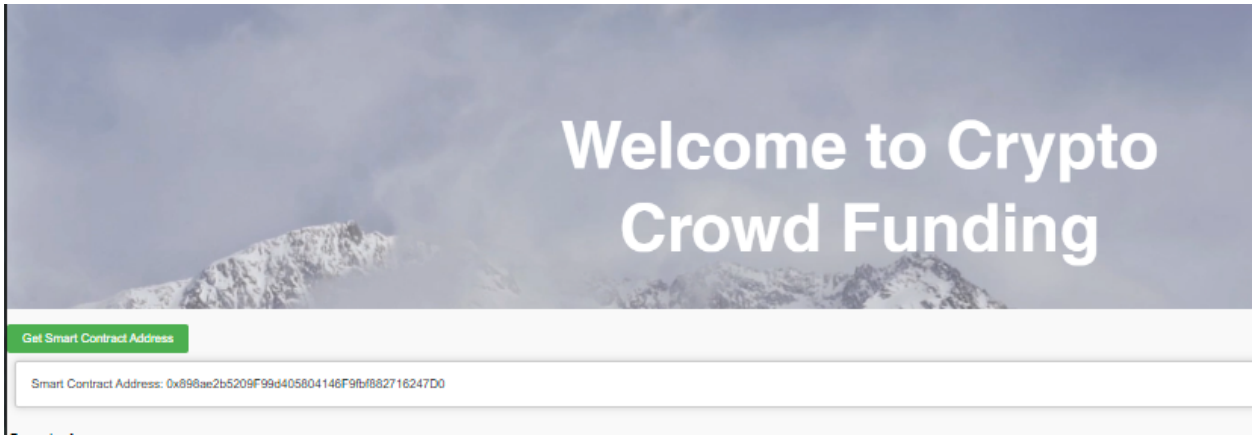
- **After you have ur wallet all setup and you have some fund because using all of these functions take some funds**
- **Now that everything is set up when u click the get smart contract address button you should see this:**

- 

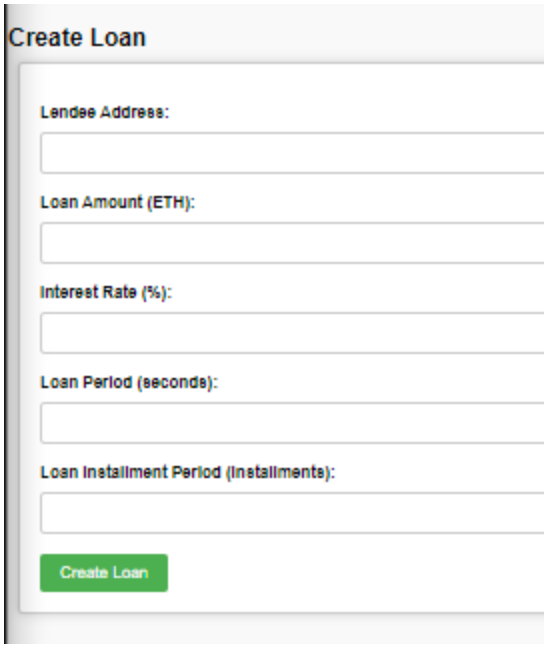
- **The getting messages should change to the smart contract address u just have to make sure u confirm the transaction to take place in the meta mask chrome plug in**



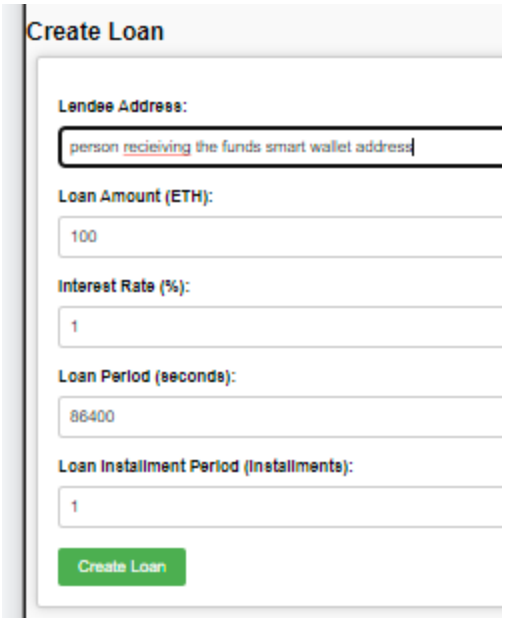
- **And then the smart contract address will show**



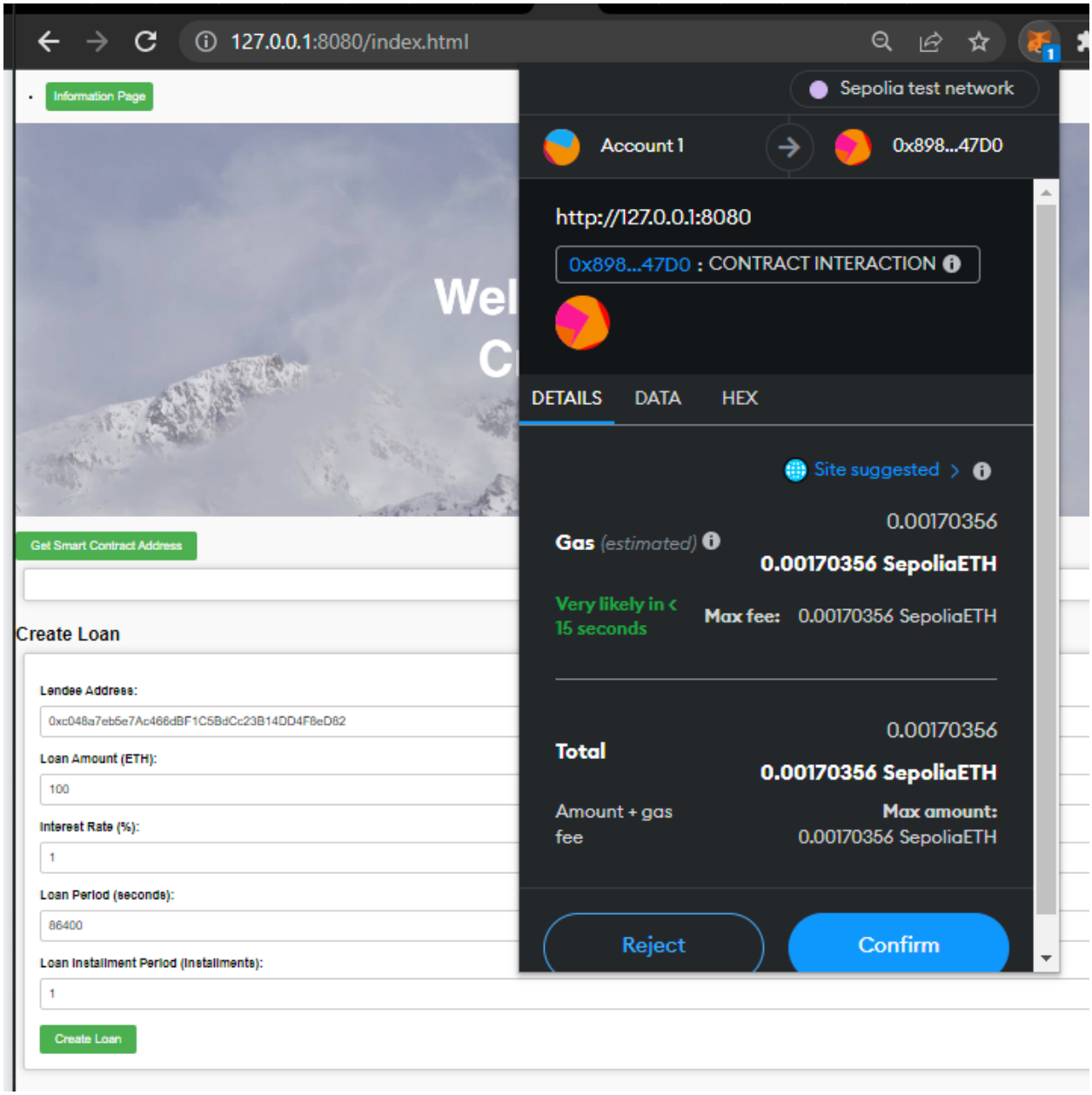
- **The next option available is to create a loan:**

- 

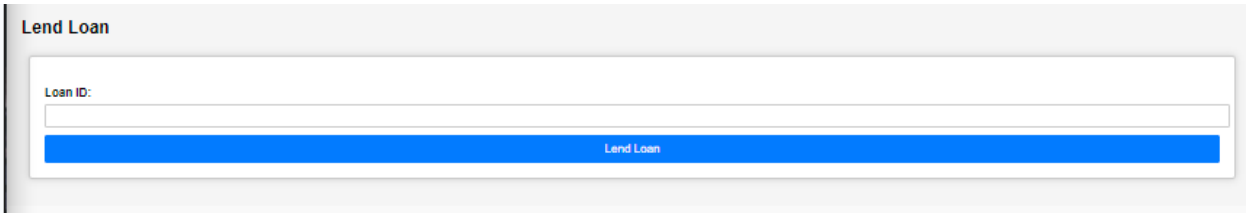
- **To use this function you must input a smart wallet address in the lendee address section, the person receiving the funds should have their smart wallet address placed there**
- **For the purpose of testing just put in your own wallet so the test eth goes back to you**
- **The loan amount is a little strange but in general u can just use 100 for testing, don't worry your not actually going to loan out 100 test eth its actually going to be much smaller than that**
- **For the interest rate for testing purposes, we just used 1%**
- **For the loan period make sure it's long, the input has to be in seconds (seconds in a day: 86400)**
- **Loan installment period for testing purposes we used 1**

- 

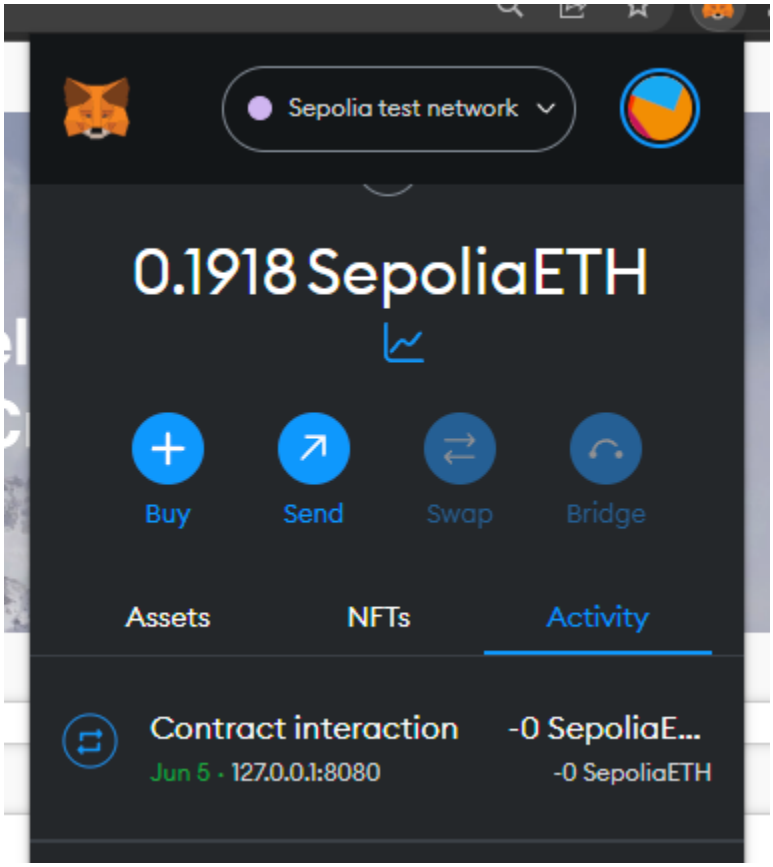
- **Then you will be greeted with meta mask asking if you want to pay for the gas to use the create loan smart contract function**



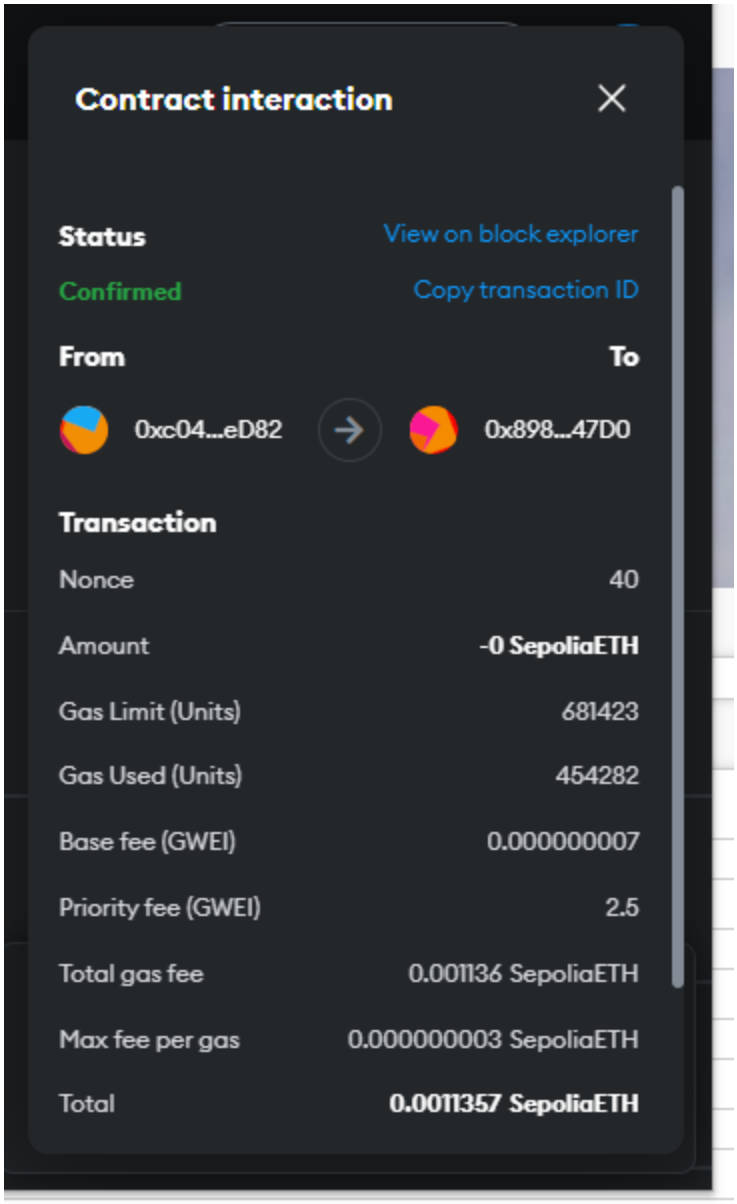
- **Disclaimer: the function below have a weird glitch where it instantly refreshes the page which ends up hiding the results**
- **Now that you have a loan made you can see its id by clicking the lend loan button**



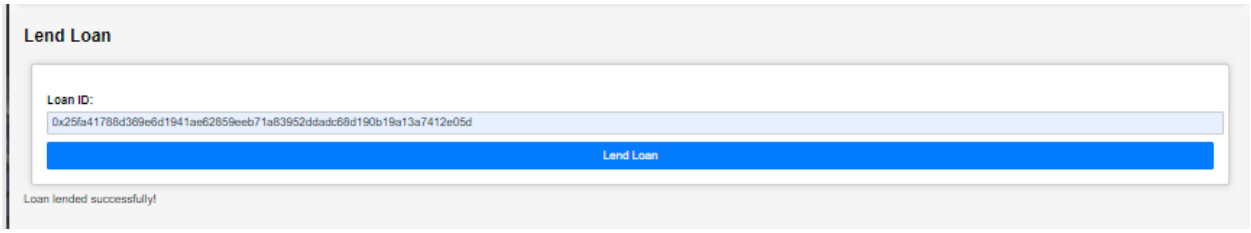
- **To get the loan id open the meta mask chrome plug-in and click on the activity tab**

- Your most recent transaction should be on the top labeled as contract interaction
- 

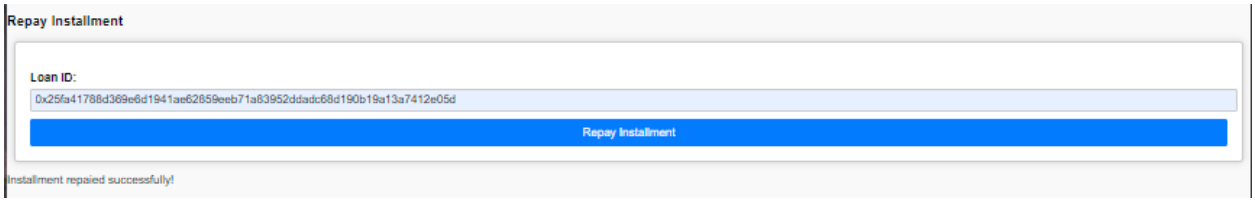
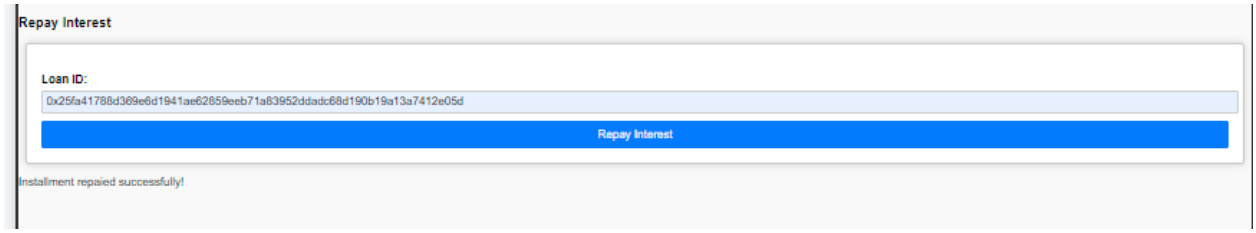
- Click on the first contract interaction that should be your most recent loan you just made
- Once you click it you will see this page



-
- Click on copy transaction id and place that in lend loan
- When you click the button you should see right underneath the section saying loan lended successfully



-
- This interaction is the same for the repay installment and interest parts just place the loan id in them aswell and click:

- 
Repay Installment
Loan ID:
0x25fa41788d369e6d1941ae62859eeb71a83952ddadc68d190b19a13a7412e05d
Repay Installment
Installment repaied successfully!
- 
Repay Interest
Loan ID:
0x25fa41788d369e6d1941ae62859eeb71a83952ddadc68d190b19a13a7412e05d
Repay Interest
Installment repaied successfully!

- **Where to take this next:**
  - **We didn't have the time to but our hope is for the front end to have isolated sections for different kinds of users depending on their goal**
  - **Implement separate pages for different kidneys of users:**
    - **We would have a section for contractors to set up projects to try to receive funding from public and private resources**
    - **A section for a common user (a singular person who wants to fund the projects)**
    - **A section for businesses (a small or large private corp can also help fund projects)**
    - **Also for the lend loan, repay installment, and repay interest when you type in something and click the button it shows a the results but then also almost immediately refreshes the page not sure why this is happening**
  - **Fix the overall look of this front end**
    - **Spacing between each smart contract function interaction on the index.html**
    - **The indenting is off as well**
  - **Implement a menu bar to navigate through the website**
  - **Most of this will be done in app.js so you have the smart contract function being called and getting the info it needs from the index.html.**
  - **Most of the smart contract functions rely on the loanID and will be simple to implement in index.html so that way users can interact with the functions.**