

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

```
let web3;
let contractInstance;
/*Connection to Toll Based Smart Contract*/
const contractABI = [
  {
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "string",
        "name": "accountWarning",
        "type": "string"
      }
    ],
    "name": "CreateAccount",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "amountToWithdraw",
        "type": "uint256"
      },
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "withdrawableRevenue",
        "type": "uint256"
      },
      {
        "indexed": false,
        "internalType": "string"
      }
    ]
  }
];
```

← ABI goes here

```
const contractAddress = '0x515ca45702925EF19a1fcb7e8392dC8b9734d131';
```

← SC address

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.returnValues[4]}  
`;
      }
    });
  } 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:

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8080'. The page has a dark header with navigation icons. The main content area features a large banner with a mountain landscape and the text 'Welcome to Crypto Crowd Funding'. Below the banner, there is a section titled 'Get Smart Contract Address' with a text input field. The 'Create Loan' section contains several input fields: 'Lendee Address:', 'Loan Amount (ETH):', 'Interest Rate (%)', 'Loan Period (seconds):', and 'Loan Installment Period (installments):', each followed by a green 'Create Loan' button. The 'Get My Active Loans' section has a green 'Get My Active Loans' button. The 'Lend Loan' section includes a 'Loan ID:' input field and a blue 'Lend Loan' button. The 'Repay Installment' section has a 'Loan ID:' input field and a blue 'Repay Installment' button. The 'Repay Interest' section has a 'Loan ID:' input field and a blue 'Repay Interest' button.

Information Page

Welcome to Crypto Crowd Funding

Get Smart Contract Address

Create Loan

Lendee Address:

Loan Amount (ETH):

Interest Rate (%):

Loan Period (seconds):

Loan Installment Period (installments):

Create Loan

Get My Active Loans

Get My Active Loans

Lend Loan

Loan ID:

Lend Loan

Repay Installment

Loan ID:

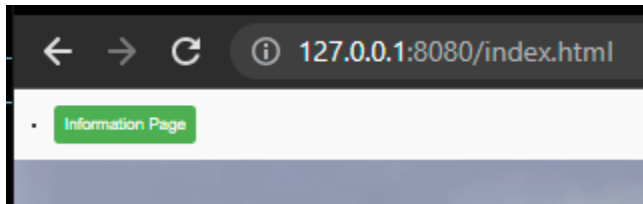
Repay Installment

Repay Interest

Loan ID:

Repay Interest

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

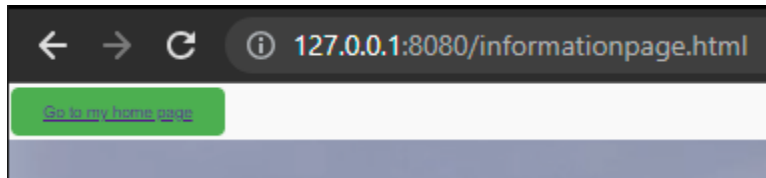


- New information page this page has basic info on our capstone project

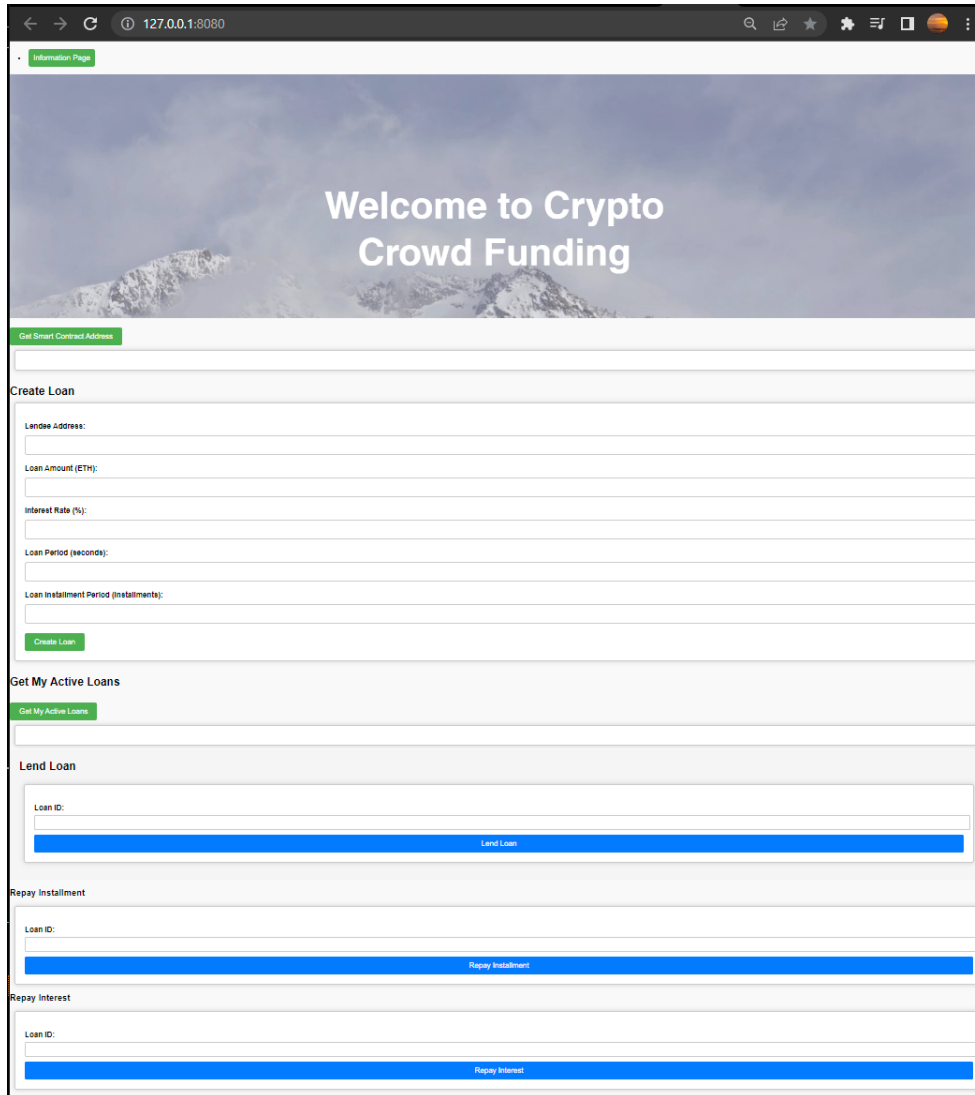
 A screenshot of a web browser showing the 'informationpage.html' at '127.0.0.1:8080'. The page features a large header image of a snowy mountain range with the text 'Welcome to Crypto Crowd Funding'. Below the header, the page is organized into several sections:

- Project Information**
 - Project Title and Personnel**: The project is called "Blockchain-based Smart Contracts for Infrastructure Funding" and was developed by Shuwen Xu, Altynbek Yermurat, Abhimanyu Bais, Derek Quoc Dang, Samuel Knox, and Michael Gadda.
 - Purpose of the Project**: Our goal is to remove delays and cost overruns during the development of a project using blockchain based smart contracts to manage funding and ownership of infrastructure assets. We want to improve transparency of transactions and allow investors to have a stake in infrastructure projects.
 - Problem Solved by the Project**: The Blockchain-based Smart Contracts for Infrastructure Funding project solves several problems faced by various audiences:
 - Borrowers**: Borrowers can access funds without the need for a traditional bank loan, which may be difficult to obtain due to stringent lending criteria or a lack of collateral.
 - Lenders**: Lenders can earn higher returns on their investments compared to traditional savings accounts or other investment options.
 - Economies**: The Blockchain-based Smart Contracts for Infrastructure Funding project promotes financial inclusion and democratizes the lending process, which can lead to a more efficient and fair economy.
 - Value Proposition**: The Blockchain-based Smart Contracts for Infrastructure Funding project offers several advantages over traditional lending options:
 - Decentralization**: The platform is decentralized, meaning that there are no intermediaries involved in the lending process.
 - Transparency**: The lending process is transparent and can be audited by anyone on the blockchain network.
 - Accessibility**: The platform is accessible to anyone with an internet connection and a supported cryptocurrency wallet.
 - Documentation and Help**: Documentation and help resources for the Blockchain-based Smart Contracts for Infrastructure Funding project can be found on the project's website, including:
 - User Guide**: A comprehensive guide on how to use the platform.
 - FAQ**: A list of frequently asked questions and their answers.
 - Support**: A support forum where users can ask questions and receive assistance.
 - Contact Information**: To learn more about the Blockchain-based Smart Contracts for Infrastructure Funding project, please contact the development team at info@lenders.com.
 - Download and Usage**: The Blockchain-based Smart Contracts for Infrastructure Funding platform can be accessed by visiting the project's website and connecting to the blockchain network using a supported cryptocurrency wallet.

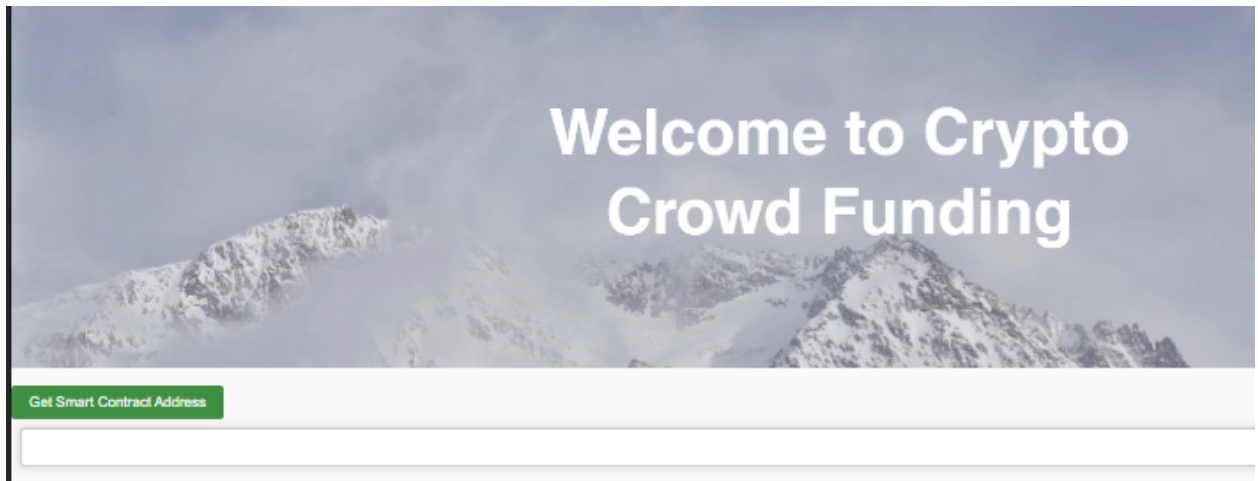
- You can click on the same button again to go back home



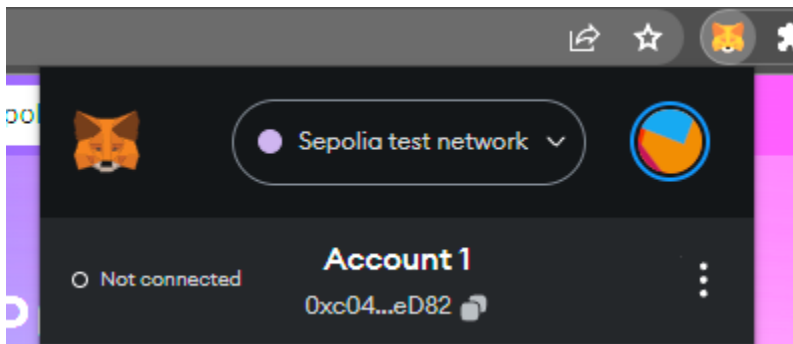
- And now your back home



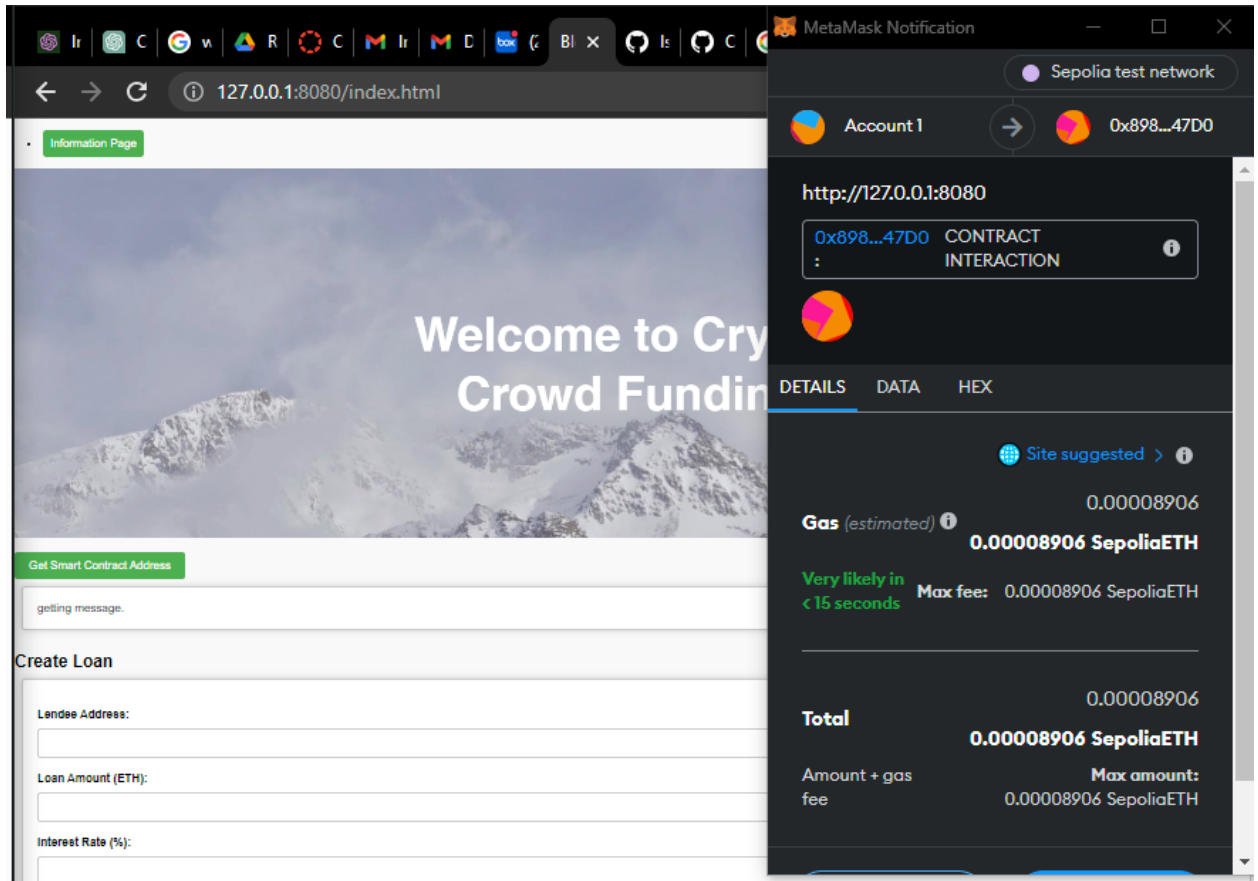
- The next button you can press and try out is the get smart contract address



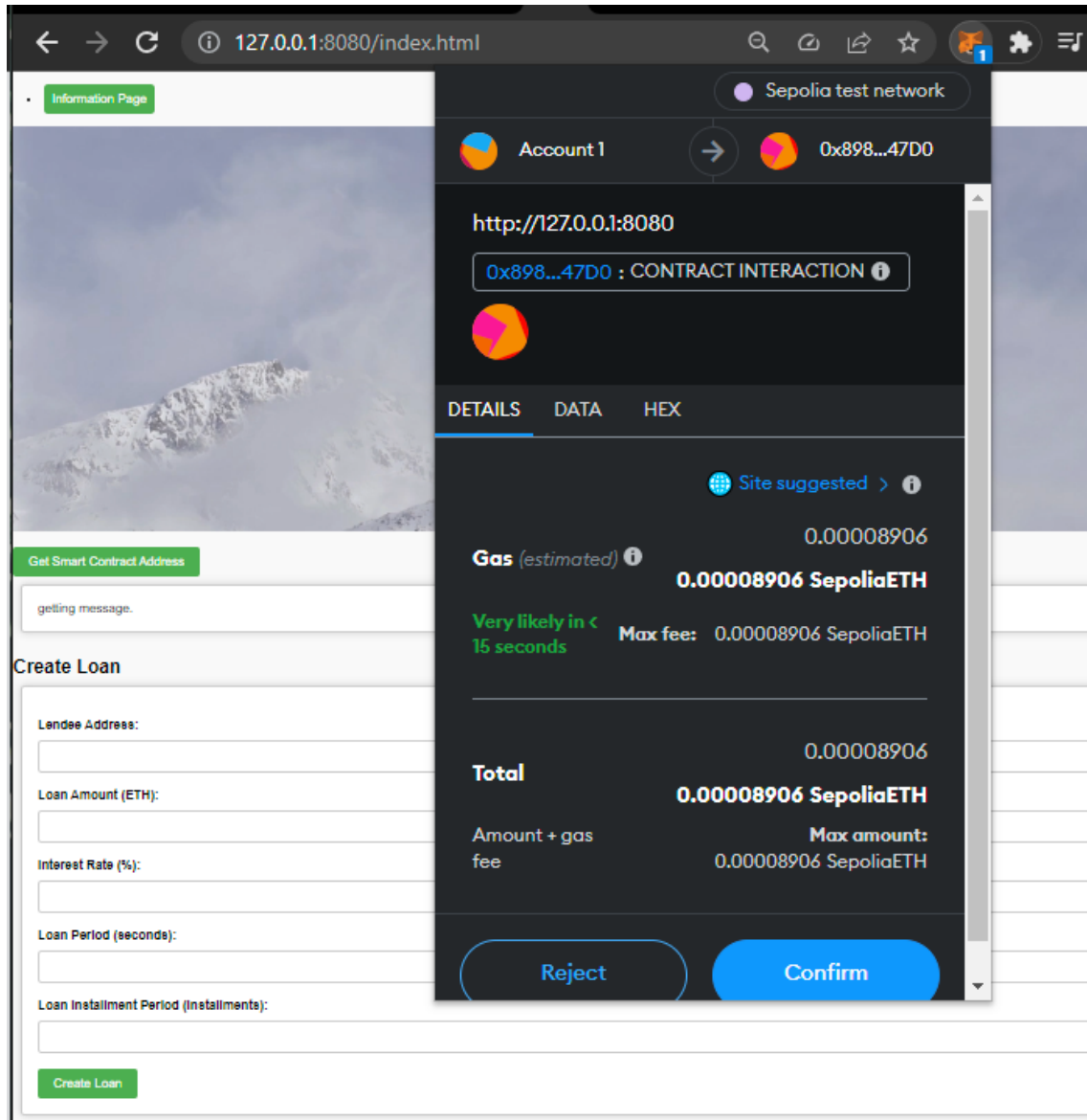
-
- There is an initial setup on meta mask where you have to set up an account I recommended for the sake of testing you ignore setting up a 12-word password setup for ur wallet but make sure that when it asks for the password you remember that
- Also, make sure you have some test eth (not actual Ethereum 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:



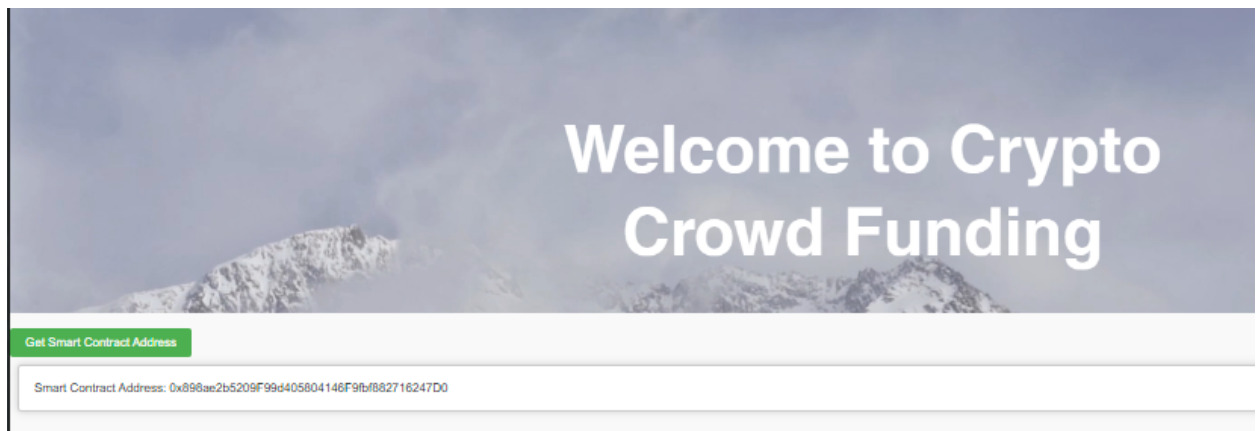
-
- After you have ur wallet all setup and you have some fund because using all of these functions take some funds
- Now that everything is set up when u click the get smart contract address button you should see this:



-
- 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:

Create Loan

Lendee Address:

Loan Amount (ETH):

Interest Rate (%):

Loan Period (seconds):

Loan Installment Period (Installments):

Create 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

Create Loan

Lendee Address:

Loan Amount (ETH):

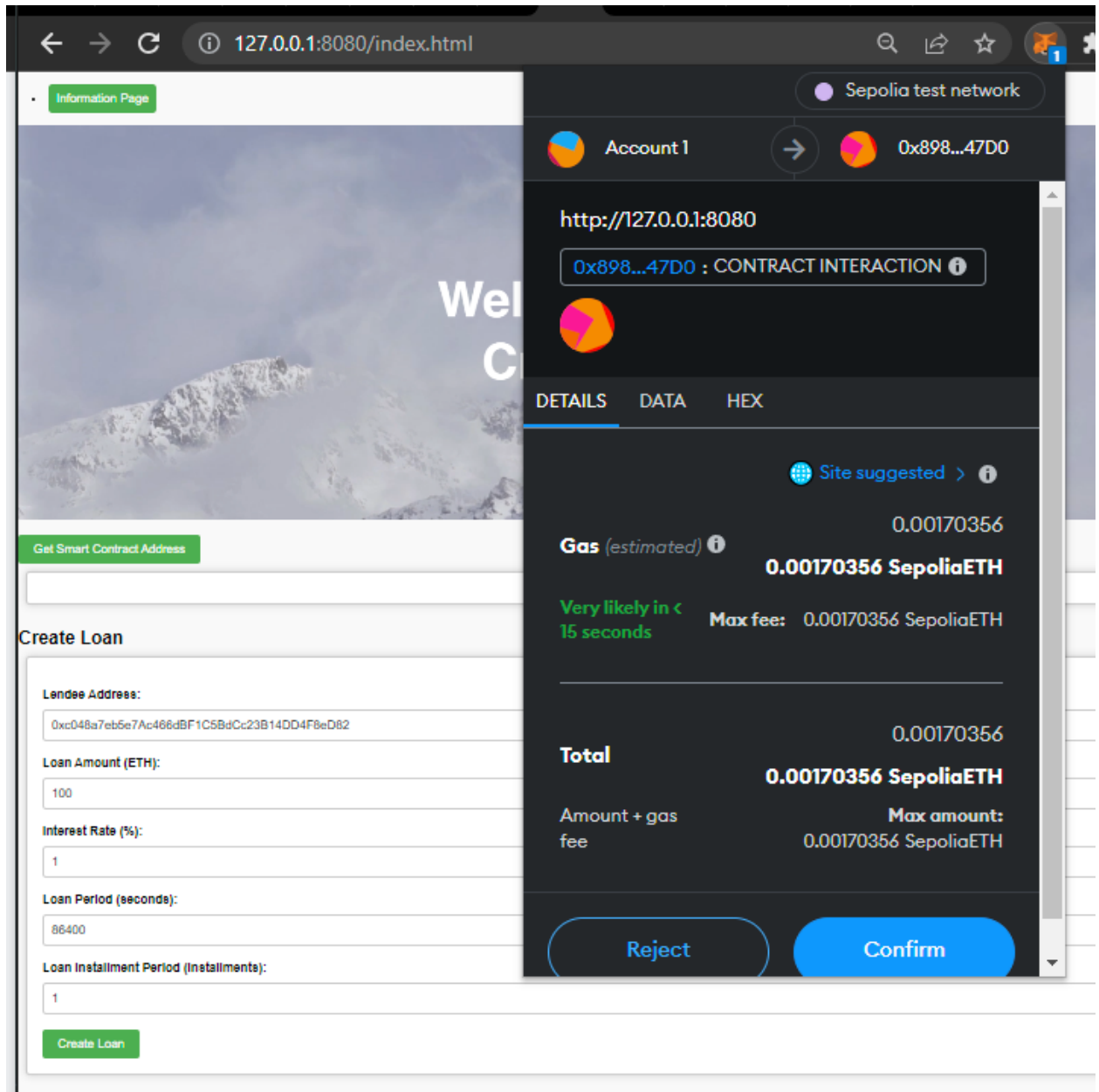
Interest Rate (%):

Loan Period (seconds):

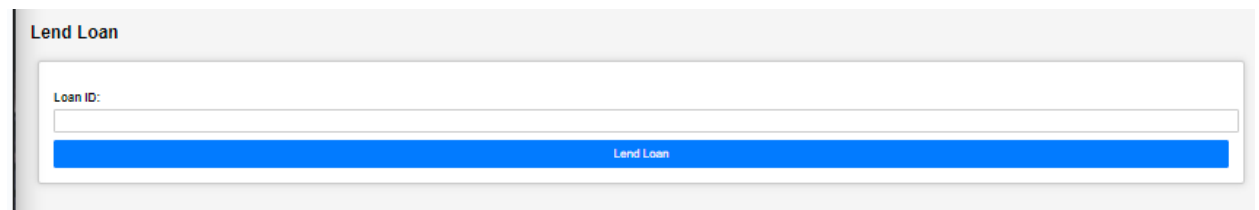
Loan Installment Period (Installments):

Create Loan

-
- 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

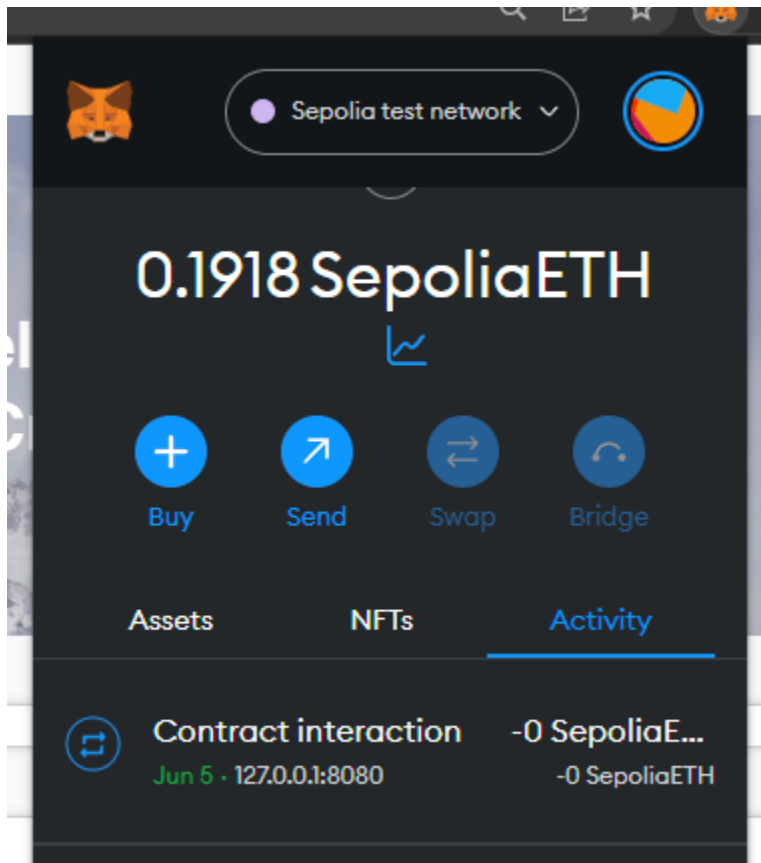


-
- Disclaimer: the function below have a weird glitch where it instantly refreshes the page which ends up hiding the results
- Now that you have a loan made you can see its id by clicking the lend loan button

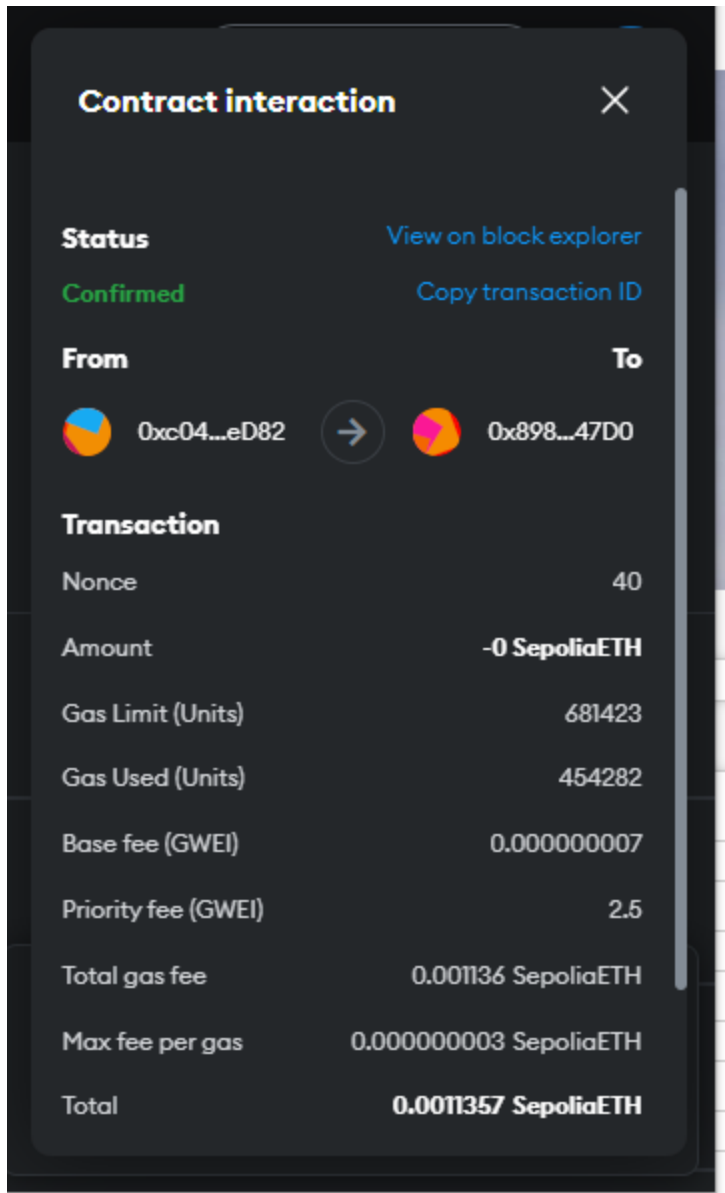


-
- To get the loan id open the meta mask chrome plug-in and click on the activity tab

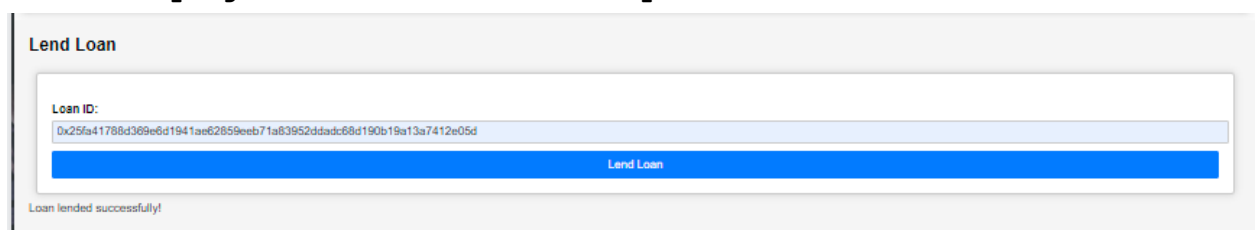
- Your most recent transaction should be on the top labeled as contract interaction



-
- 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 repaid successfully!

Repay Interest

Loan ID:

0x25fa41788d369e6d1941ae62859eeb71a83952ddadc68d190b19a13a7412e05d

Repay Interest

Installment repaid 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 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.