The Bounty App written on this project is divided into two main parts, the smart contract and the user interface, which are fully explained in the following sections

* **Smart contract:**

In this project, we have tried to define the required variables first. Then it's time to define events and then modifiers. Also, at the end, the required functions of this application are written, which are examined separately for each one.

**Variables:** In this contract, there are variables with data types int, address, structure and mapping. The UserID variable is the user ID. The JobID variable is the identifier of each job. UserProfile variable is a structure related to each person that is visible in the user variable depending on the user ID for each person. Similarly, the JobProfile variable is a structure related to each job, which can be identified in the job variable according to the job ID for the job that was created. The reason for using mapping is to use the key = value property of this variable. Also, a variable with the type of address is considered for the owner of the smart contract, which is defined as private due to security reasons, so that it can be accessed only in the contract itself. Figure 1 shows the desired contract variables.

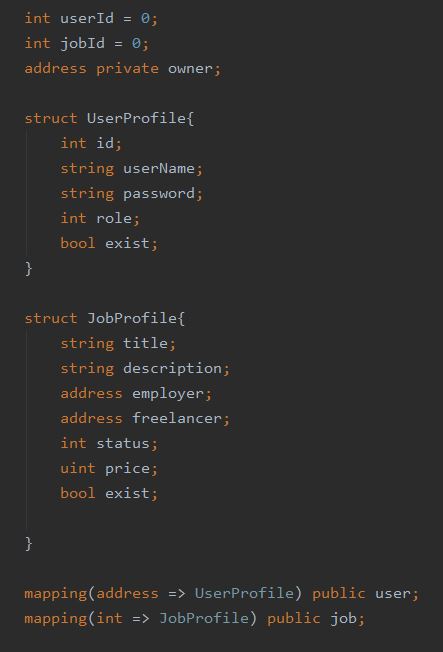
****

Figure1. Variables

**Events**: In this contract there are three events. One event to display the completion of work, another for user login and the last for job creation. Event names are jobIsFinished, userIsLogin and jobIsCreated which can be seen in Figure 2.

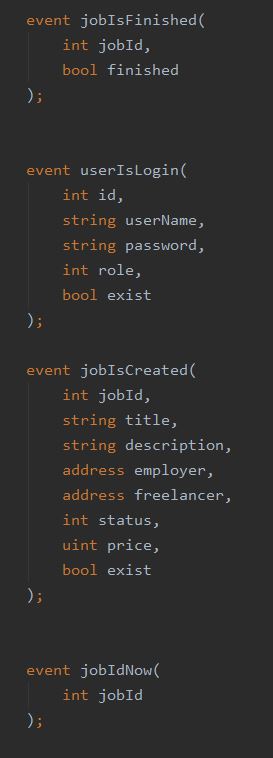
****

Figure2. Events

**Modifiers:** In this contract we use six modifiers to control different items.

1. **IsOwner**: To control that the sender of the message for some functions (such as deleting the contract) is the contract owner. Other people cannot execute these functions.
2. **isDuplicated**: To check that the user who registers in the system has not previously registered with his account address, otherwise a message will be displayed to the user to prevent re-registration.
3. **isRegisteredUser**: The user's ID and password are checked when logging in with the username and password entered during registration so a user with the same address cannot access to another user's page.
4. **isValidRole**: When the user wants to create a job, it had to be controlled that role be of the employer type.
5. **IsValidJob**: When multiple requests are made to change the desired job status, this modifier checks the identity and current status of the desired job so the process does not change in the event if there is a problem.
6. **isValidTransfer**: When transferring money, it controls that the user who deposits the money is the same employer related to the job. It also compares the amount of money in the employer's account with the amount of the job fee so check that the employer has enough money to pay.

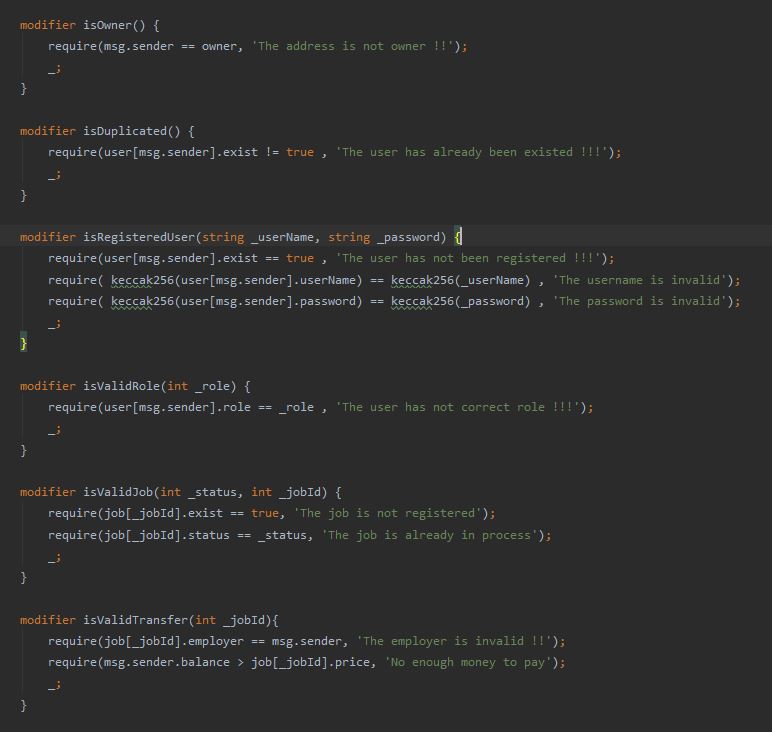


Figure3. Modifiers

**Constructor**: In the constructor, the address of the sender of the smart contract is stored as the address of the owner of the smart contract.

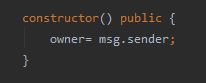


Figure4. Constructor

**signUp:** To register users, a function(its name is signup) is written.It takes the input values of user such as name, password and role. The role can be an employer or a freelancer. If the role of the user is the employer, the number 0 and if it is a freelancer, the number 1 is assigned to the role variable for each user. Also, event information about the user is called, which is used in the user interface. IsDuplicated is used to control registration.

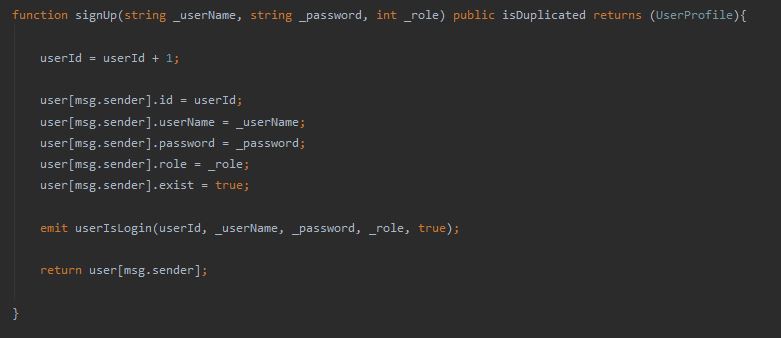
****

Figure5. Sign up function

**login:** This function is written for login, the input values are user name and password. In this function, information about the user is called using the event that is used in the user interface. IsRegisteredUser is used to control user login.

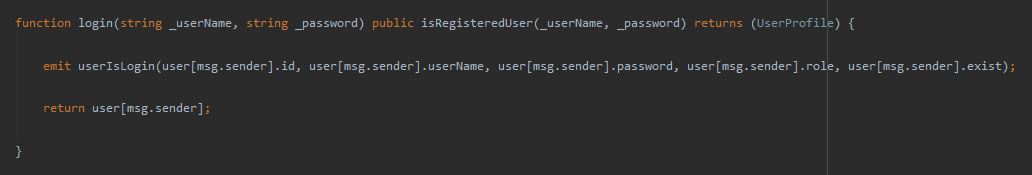
****

Figure6. Login function

**createJob:** To create a job, the employer uses a function called createJob. Its input values are title, description, status and price. Each job can have 4 status. When job is defined, its status is 0.

The status value will be 1 when the freelancer applies for the job. The status value will be 2 if the employer approves the freelancer to do the job. The status value is 3 when the freelancer completes the request. If the employer transfers the desired cost to the freelancer, the status will have a value equal to 4. In this function, job information is called using the event that is used in the user interface.

isValidRole is used to control the role of the job-creating user who must be the employer. Also in this function, the employer's account balance is controlled with the amount that is considered for the job so that he cannot allocate more than his account to the project.

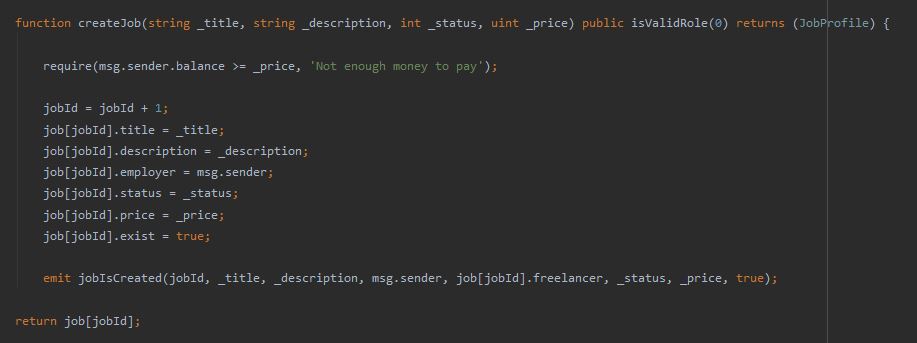
****

Figure7. Create job function

**getJobId**: This function is to get the latest ID created for jobs. This function is used in the user interface to display all the tasks that have been done so far. The return value of this function is a number.

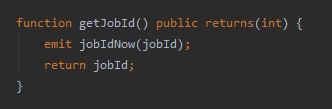


Figure8. Get job Id function

**getJob**: This function is to get information about each job based on the value of the jobId input. This function is used in the user interface to receive information about all the tasks which created in the smart contract. The return value of this function is the information about the job.

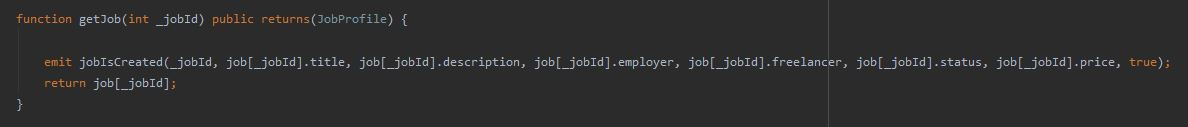


Figure9. Get job function

**confirmByFreelancer**: After the job is created by the employer, it is time to apply for a freelancer to do the desired job. In this function the role of the user who makes this request is controlled by using the isValidRole and isValidJob modifiers. Also, if the mentioned items are confirmed, the status related to the desired job will be changed from 0 to 1.

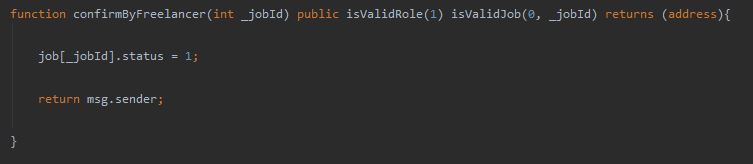


Figure10. Confirm by freelancer function

**confirmByEmployer**: After the freelancer requests a job, it is time for the employer to approve the freelancer to do the job. In this function, using the isValidRole and isValidJob modifiers, the user role that makes this request is created, the job identity is created and the status of the job is controlled. On the other hand, the address of the employer confirming the freelancer is also checked by the address of the creator of this job so that no other employer can confirm this request. If the mentioned items are confirmed, the status related to the desired job will be changed from 1 to 2 and the freelancer address will be assigned as the address related to the freelancer performing this job.

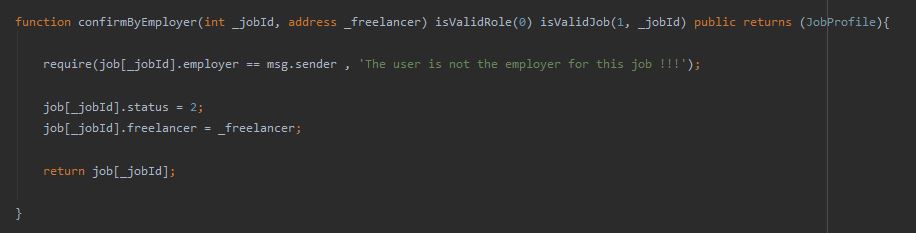


Figure11. Confirm by employer function

**workSubmission**: After the employer approves the desired freelancer, the freelancer does the work and then requests the completion of the work. The workSubmission function is used. In this function, using the isValidRole and isValidJob modifiers, the user role that makes this request, the job identity and the status of the job is controlled Also the address of the freelancer of the applicant will be checked with the approved address for the freelancer of this job so that no other freelancer can submit this request. If the above items are confirmed, the status of the job will change from 2 to 3.

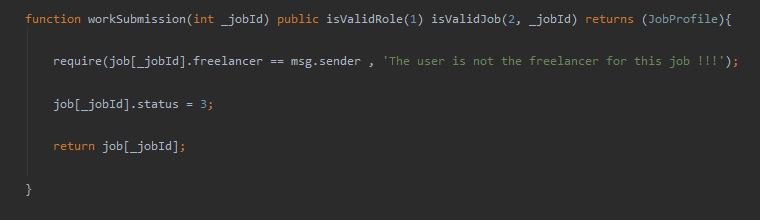


Figure12. Work submission function

**transfer**: After the freelancer requests the completion of the work, the employer can transfer the project amount to the freelancer account. The transfer function is used for this purpose. In this function, the details of this payment are checked using isValidTransfer, and if these items are confirmed, the job cost will be transferred to the freelancer account, and the status of the desired job will be changed from 3 to 4.

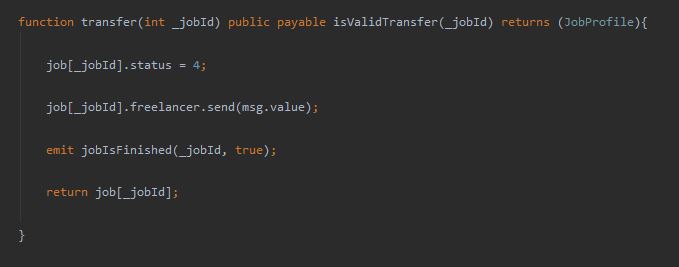


Figure13. Transfer function

**remove**: This function is used to delete a smart contract which is only enabled by the contract creator.



Figure14. Remove function

* **User interface:**

In this project, React has been used for user side programming. Figure 15 shows the structure of the files. In this structure, user-side codes are stored in the client folder. The project structure is also truffle.

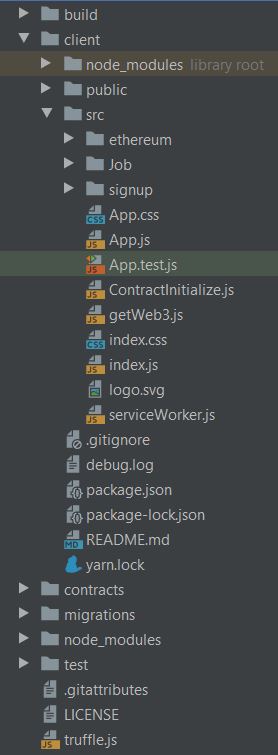
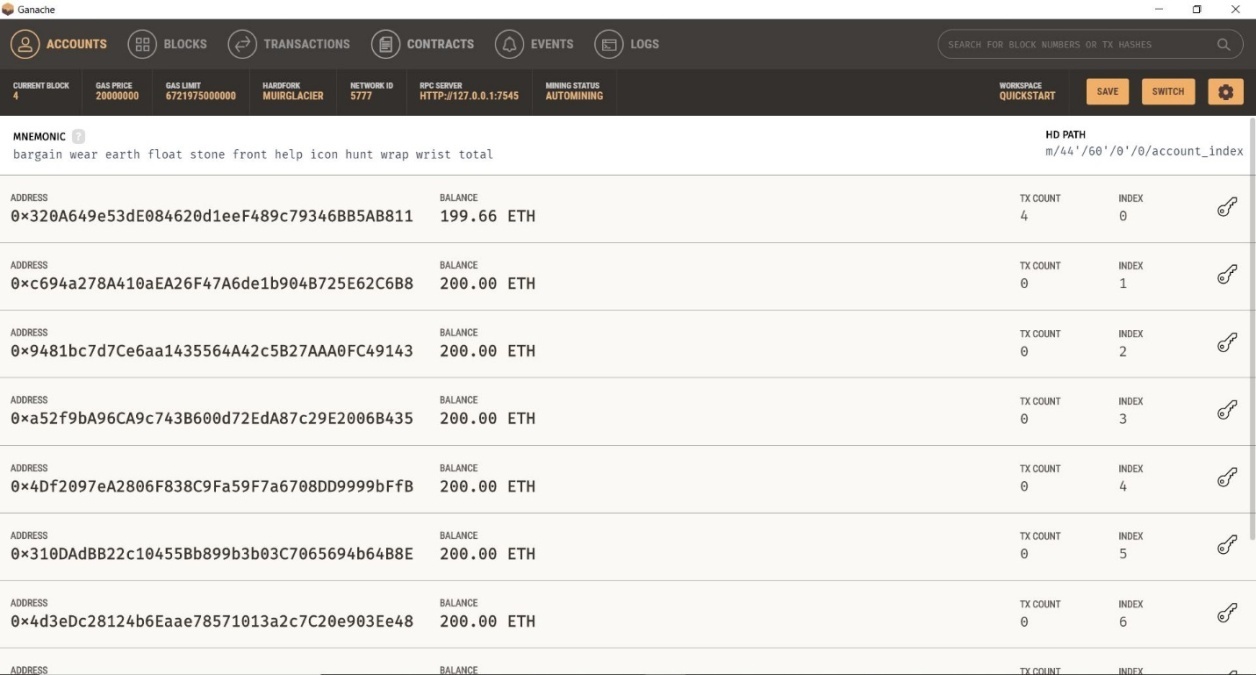
****

Figure15. Project structure

In this project, ganache is used as a testnet. Figure 16 shows the testnet for this project.

****

**Figure16. Ganache environment**

After opening the ganache environment, it is time to run the project and migrate it to truffle. To do this, we use the truffle migrate command code in the project directory. Figure 17 shows the execution of this command and the migration of the smart contract in ganache.

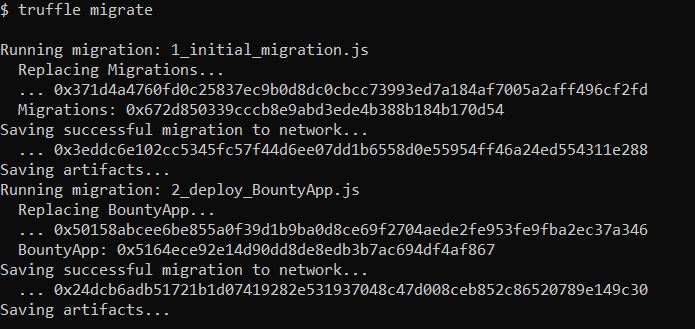
****

Figure17. Truffle migrates

At this stage we need to run the project using the command code npm run start. After running in the browser, a message appears stating the metamask link installed in the browser and ganache (Figure 18). To do this, click on the” import using account seed phrase” icon. By clicking on this page icon, we will enter a new page where we can enter the desired wallet seed.

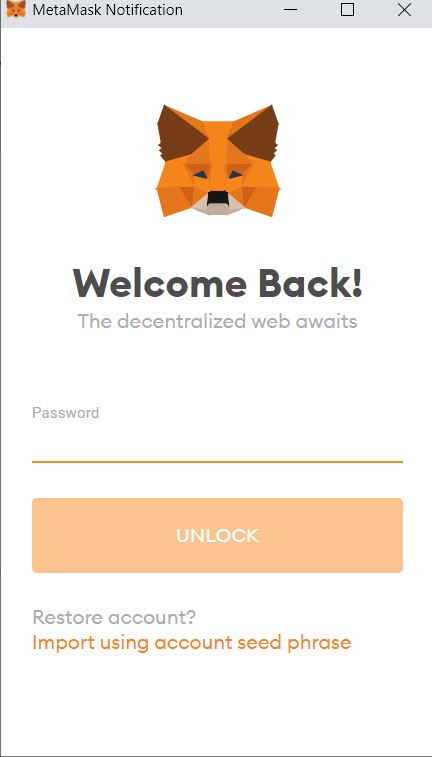


Figure18. Metamask notification

The wallet seed intended for this testnet is the same mnemonic found in ganache (Figure 19).



Figure19. mnemonic string

Also, by defining a password, we can access this testnet created in ganache (Figure 20).

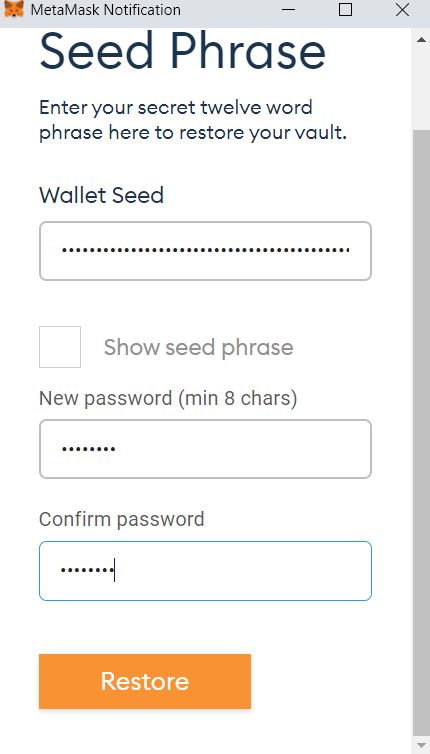


Figure19. Wallet seed

By registering the information required to communicate, the required accounts need to be associated with the desired domain, which we have used in this project using the select all option (Figure 20).

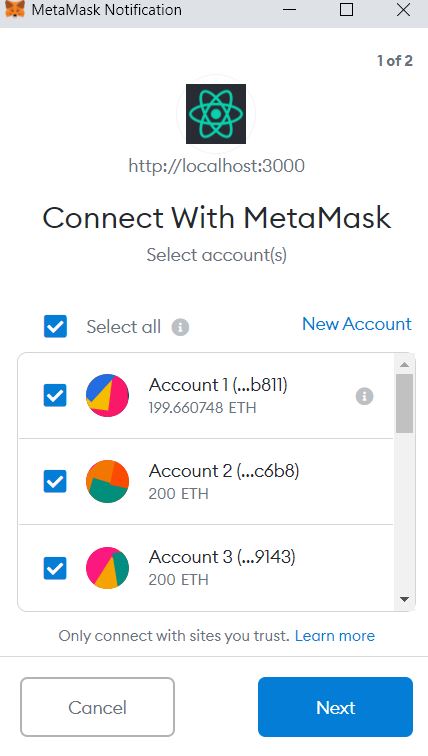


Figure20. Link accounts

By creating a connection between ganache and metamask, the created smart contract functions can be used.

The first page that the user encounters when entering the site is the sign up page (Figure 21).

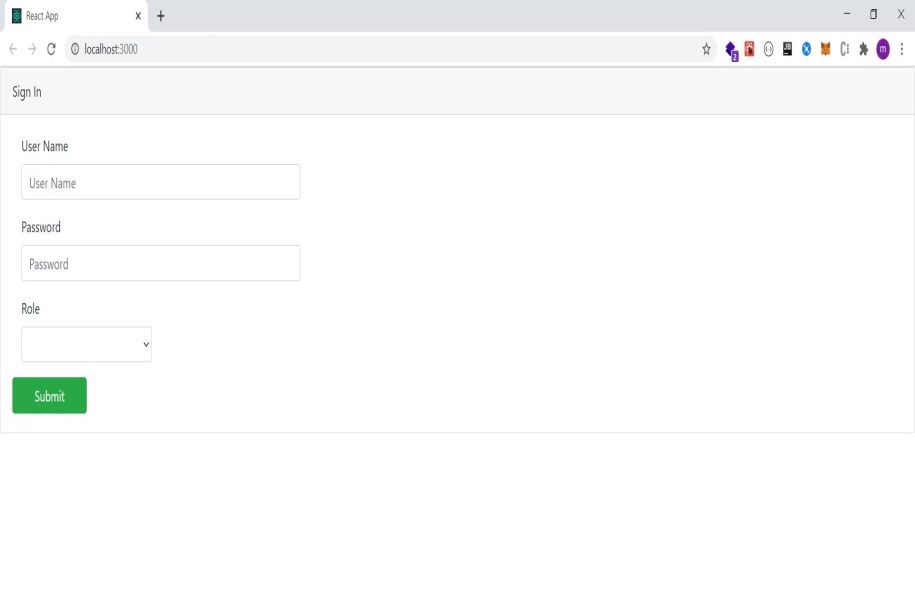


Figure21. Sign up page

By entering the required information in the signUp function, it can be called and the user registers on the site (Figure 22).

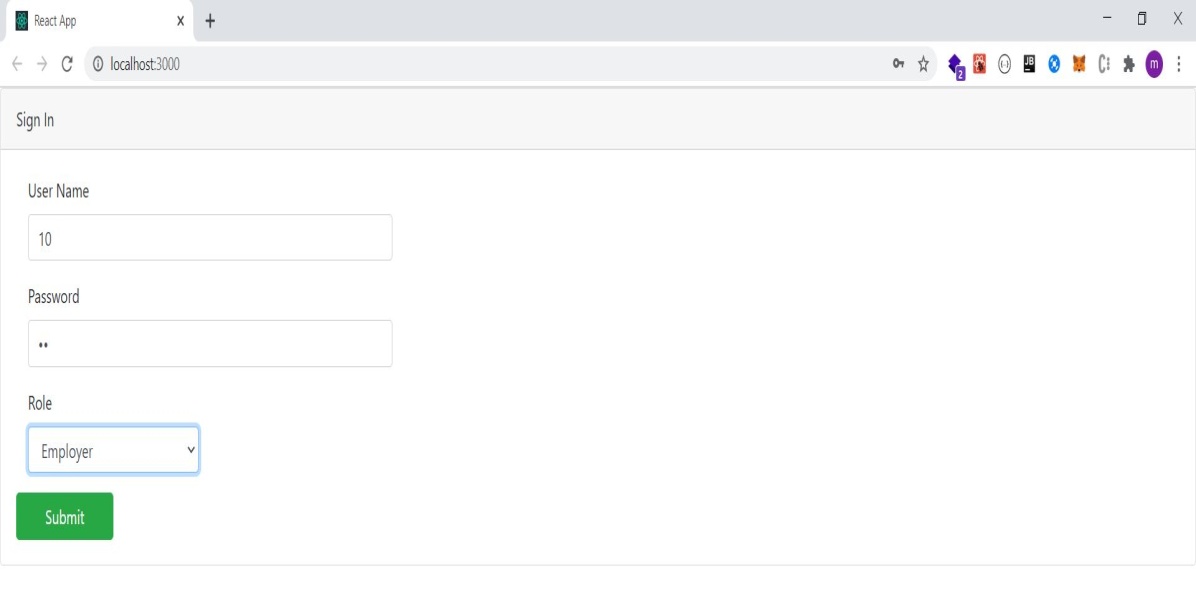


Figure22. Fill information to sign up

To call a function, the account who called that function needs to confirm the transaction in its browser metamask (Figure 23).

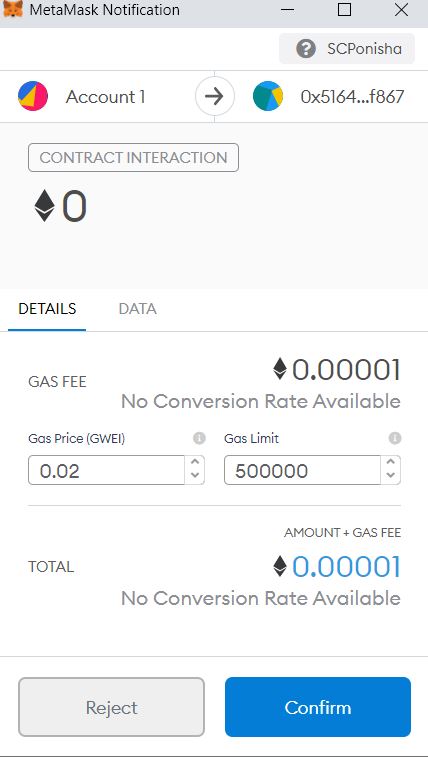


Figure23. Confirm transaction to sign up

By confirming this transaction, we enter the login page (Figure 24).

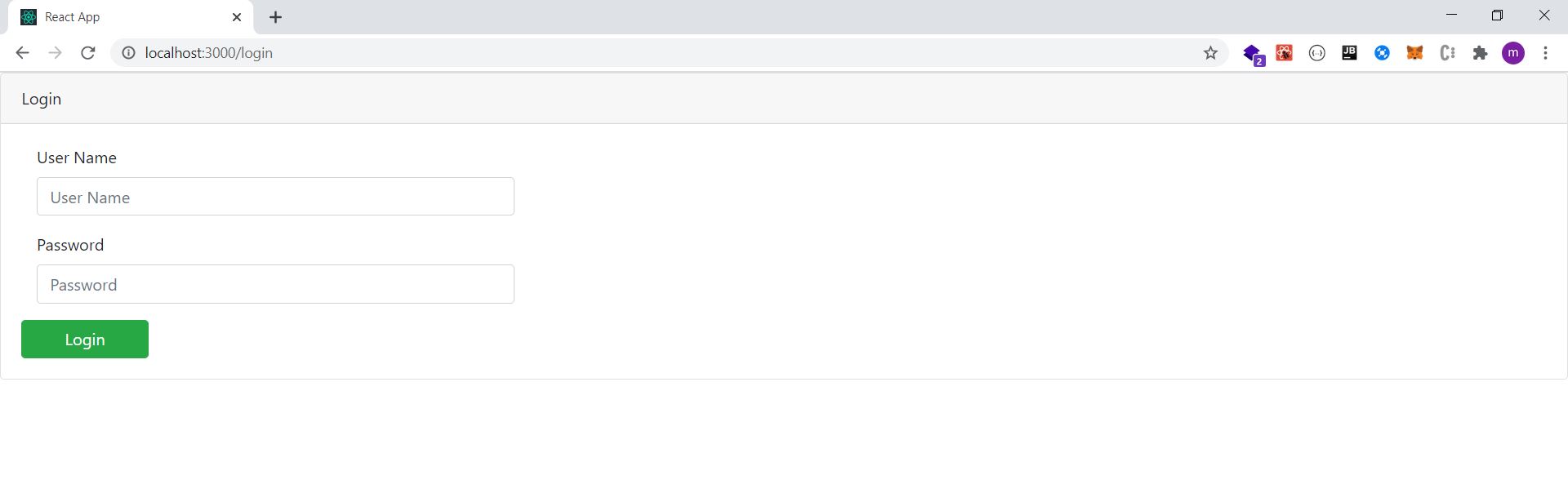
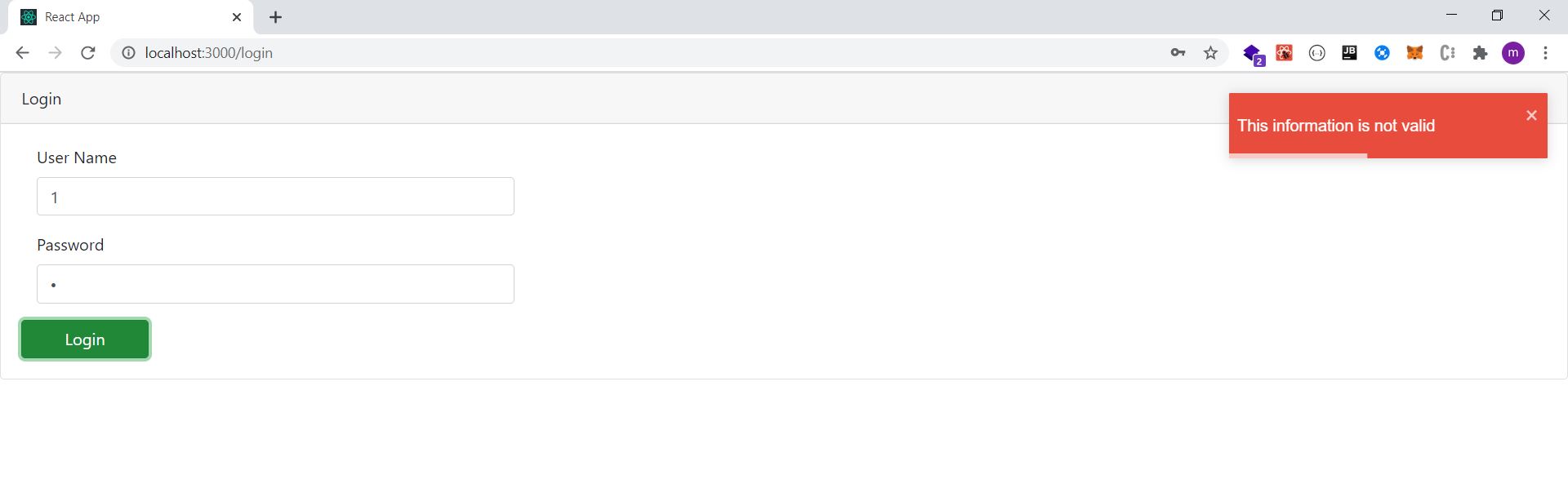


Figure24. Login page

By entering the username and password on this page, the user can enter the jobs section. If the user input information does not match the information at the time of registration, the login function written in the smart contract will not run and will display an error stating that the information is invalid (Figure 25).



**Figure25. Incorrect information**

After importing the correct information, the user can enter the jobs page by confirming the transaction (Figure 26).

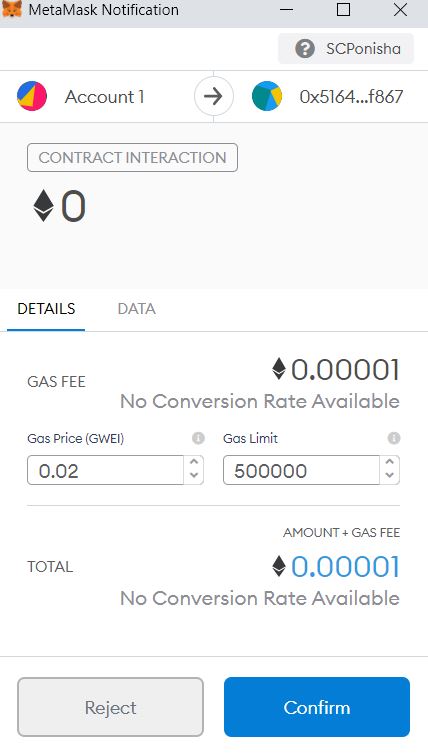


Figure26. Confirm transaction to login

The jobs page contains two buttons to create a new job and display the jobs created in the smart contract (Figure 27).

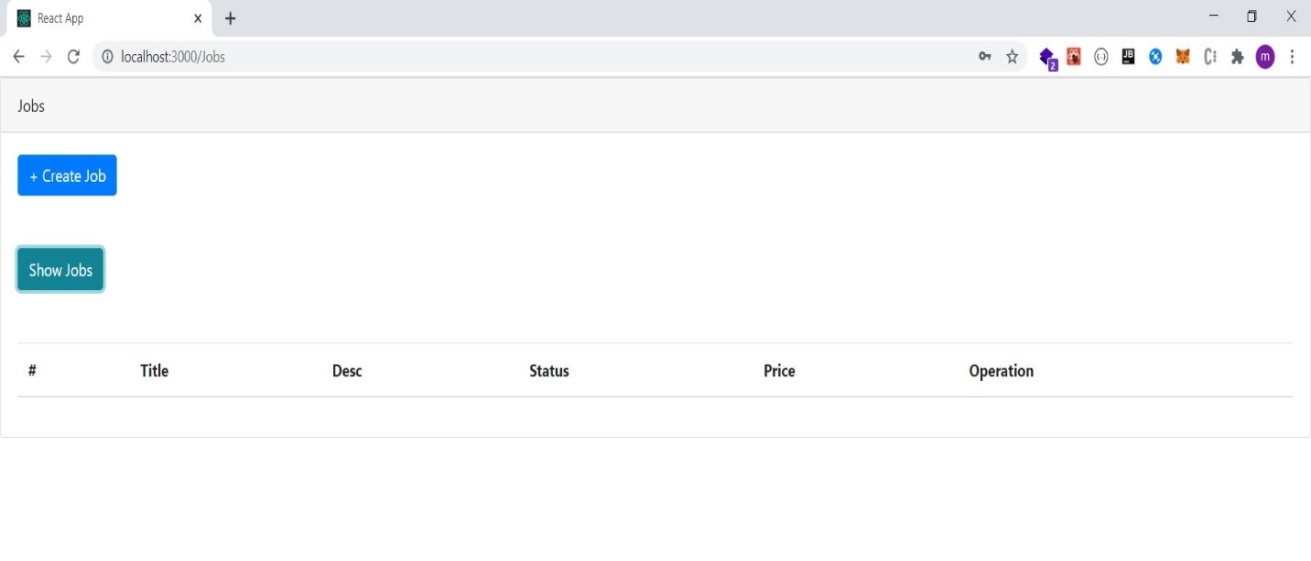


Figure27. Jobs page

By clicking the create Job button, we enter the create job page where new jobs can be created (Figure 28).

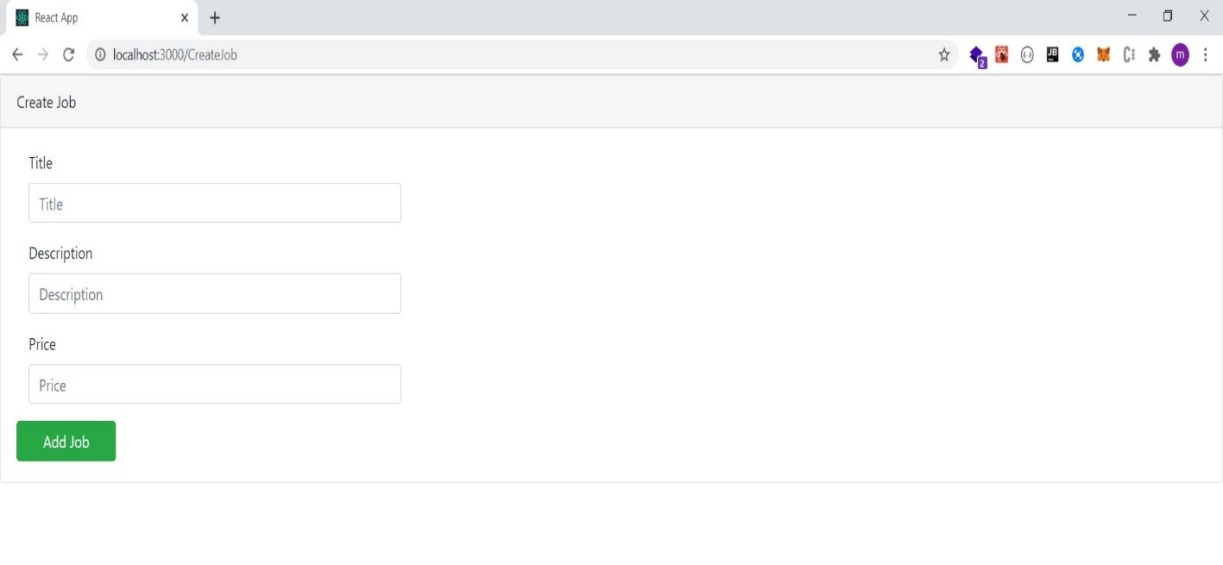


Figure28. Create job page

On this page, a job can be created by registering the required information and clicking the Add Job button (Figure 29). This is done by confirming the transaction related to this function.

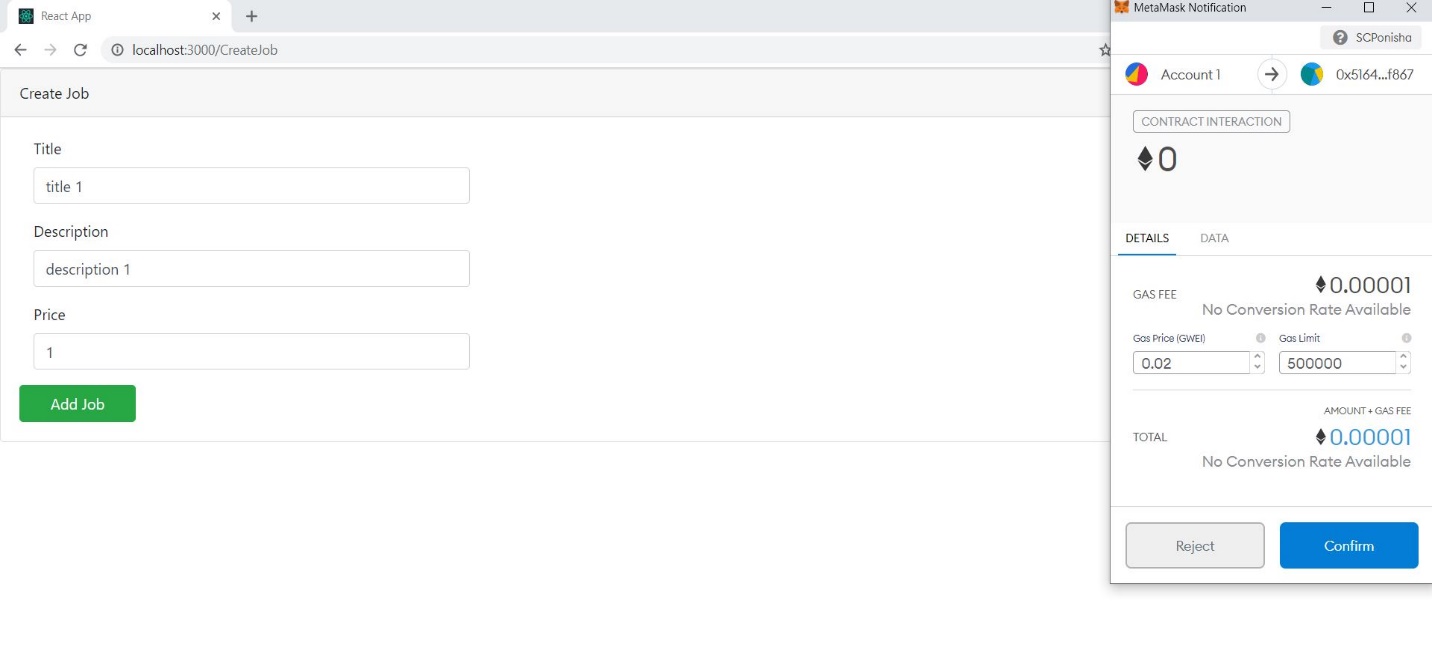


Figure29. Create job transaction

By creating new jobs, we return to the jobs page. On this page, by clicking the show jobs button, you can see the jobs created by users.

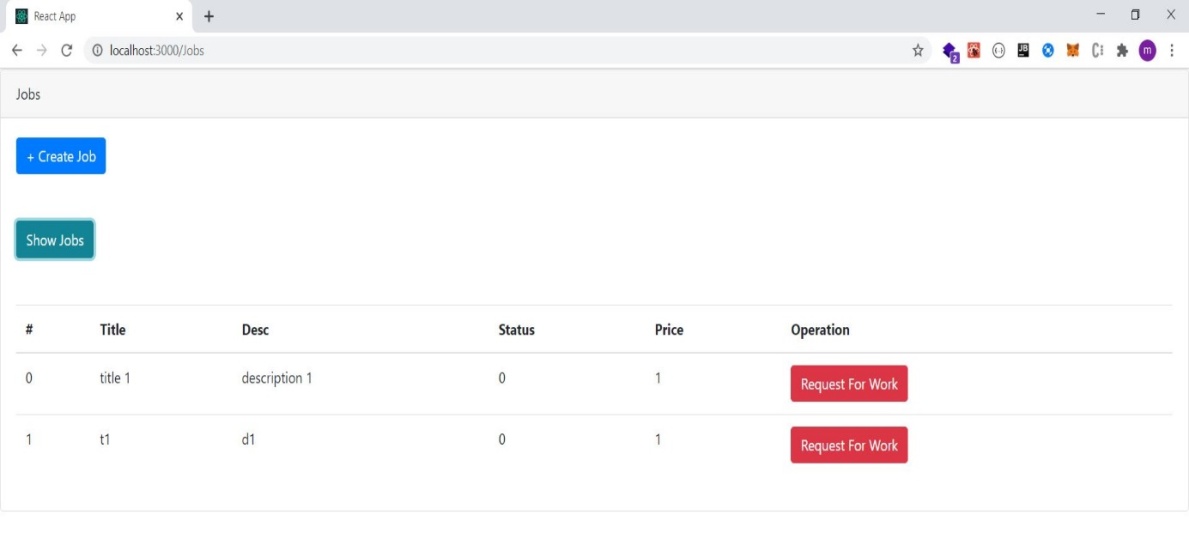


Figure30. Show jobs

In the next step, the freelancer user needs to apply for the desired job. On the other hand, according to the modifiers written for this function, a user with the role of the employer cannot do this. For this reason, it is necessary to perform all the steps related to the registration and login of the user with the role of freelancer to this page with the second account connected to metamask. Figure 31 shows the change of account from 1 to 2 in the metamask plugin.

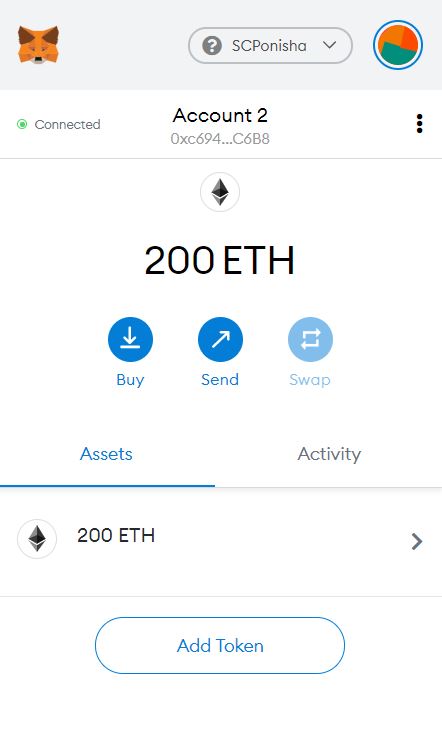


Figure32. Change account to sign up as a freelancer

Figure 33 shows the user registration with the role of freelancer.

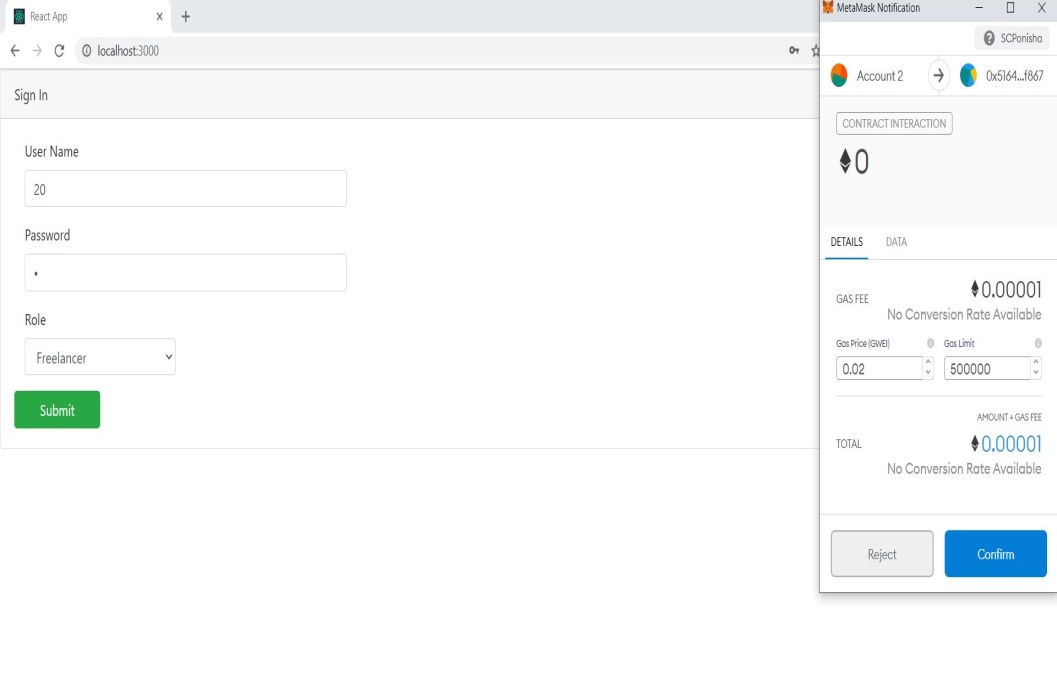


Figure33. Sign up as a freelancer

Figure 34 shows the user login as a freelancer.

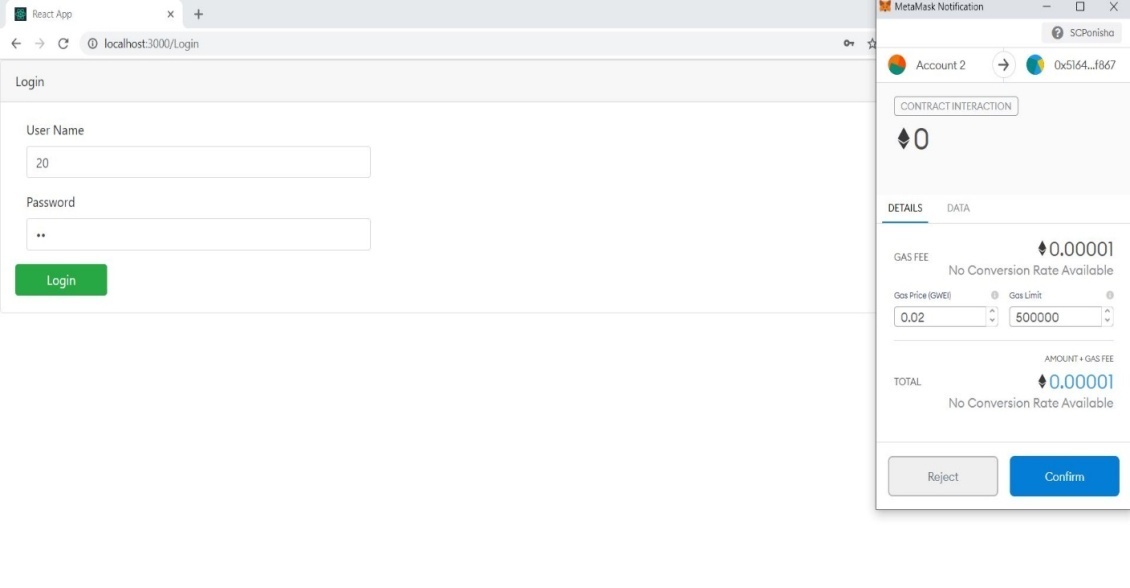


Figure34. Login as a freelancer

After the freelancer enters the jobs page, the user can apply for the desired job, which is done by the user confirming the transaction.

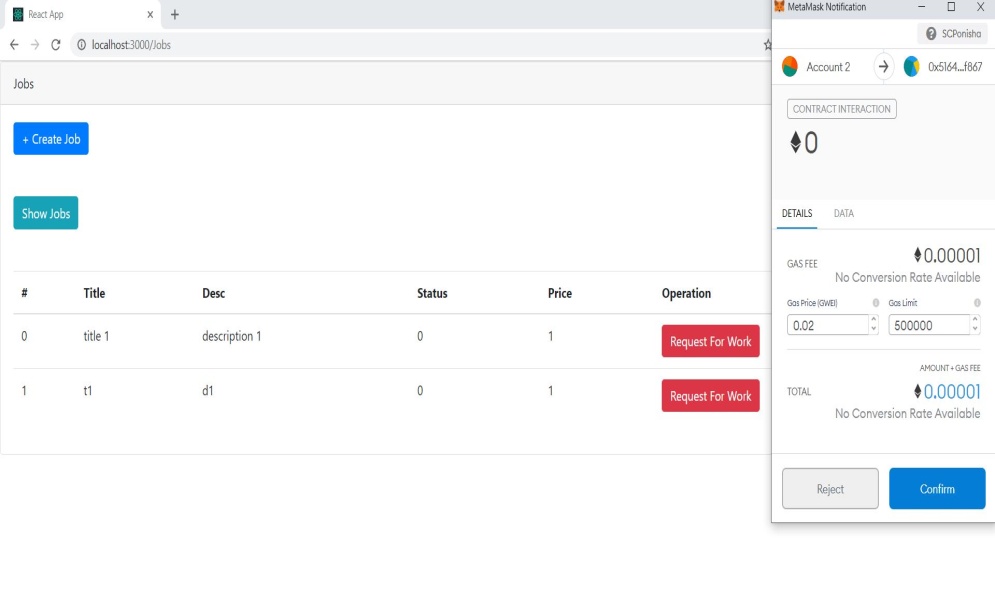


Figure35. Request for work by freelancer

Upon confirmation of the transaction, the job status is transferred from 0 to 1 and the freelancer confirmation by the employer button is displayed (Figure 36).

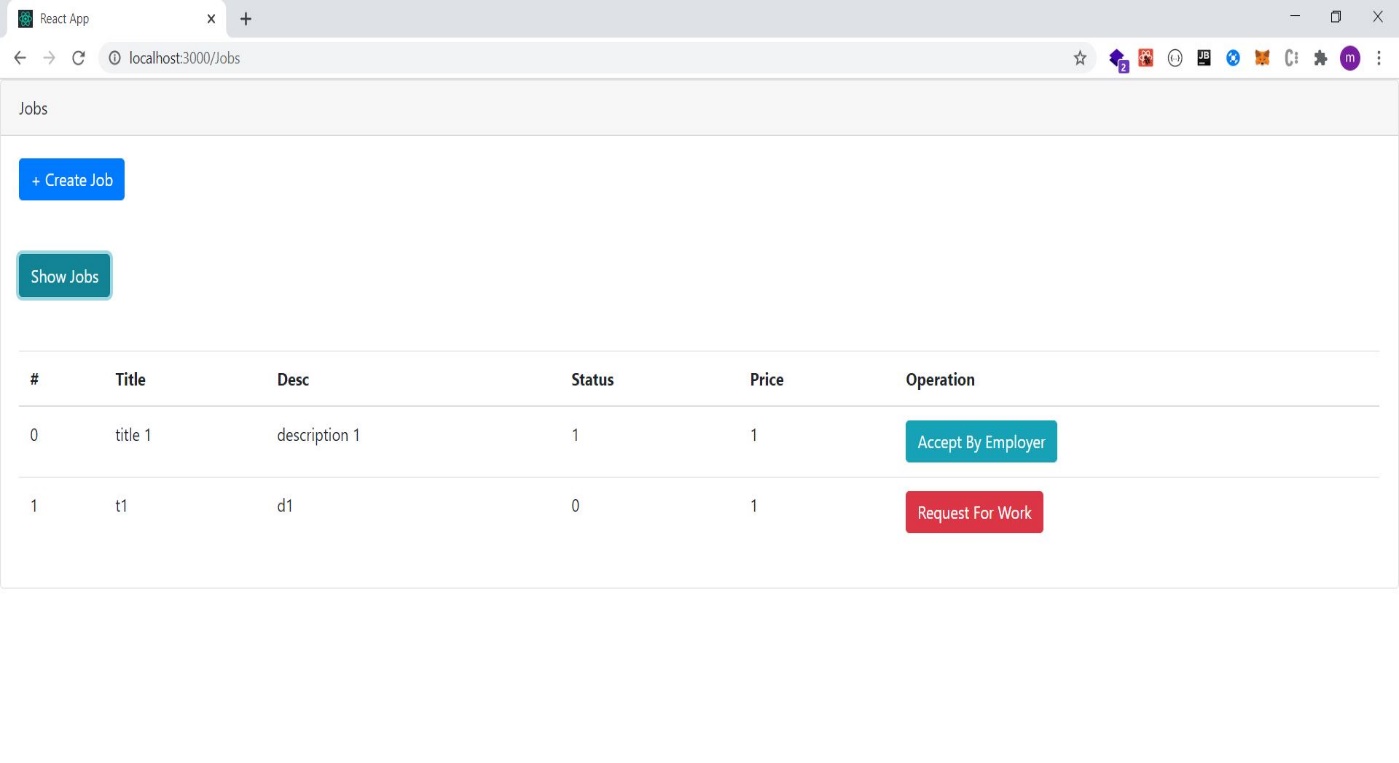


Figure36. Change status to accept by employer

To be approved by the employer, this needs to be done with the employer's user account. That is why we apply this change in metamask (Figure 37).

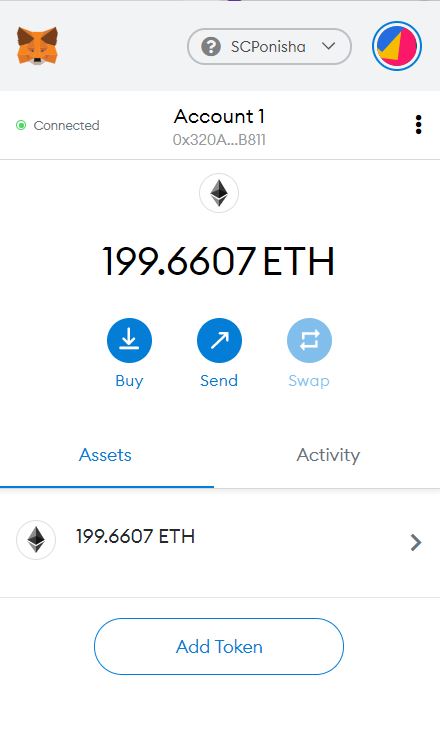


Figure37. Change account to accept freelancer as a employer

The employer user confirms the desired freelancer by clicking the accept by employer button (Figure 38).

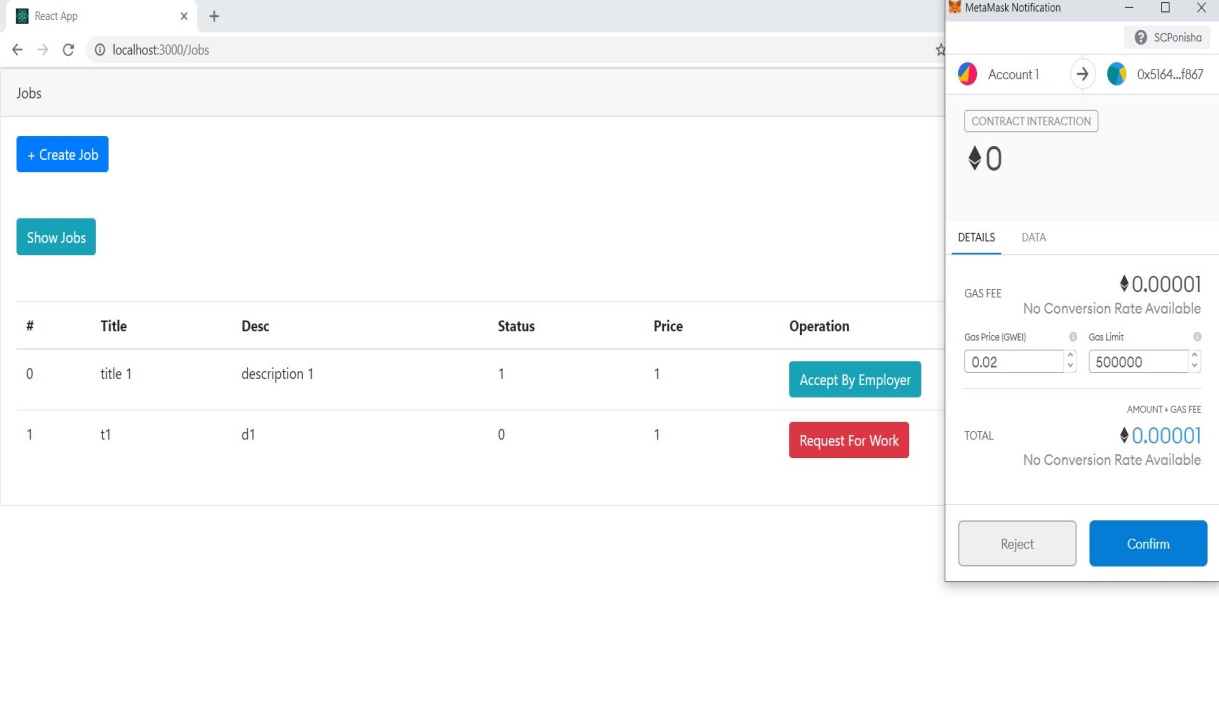


Figure38. Accept by employer transaction

By pressing the accept by employer button, the job status changes from 1 to 2 (Figure 39).

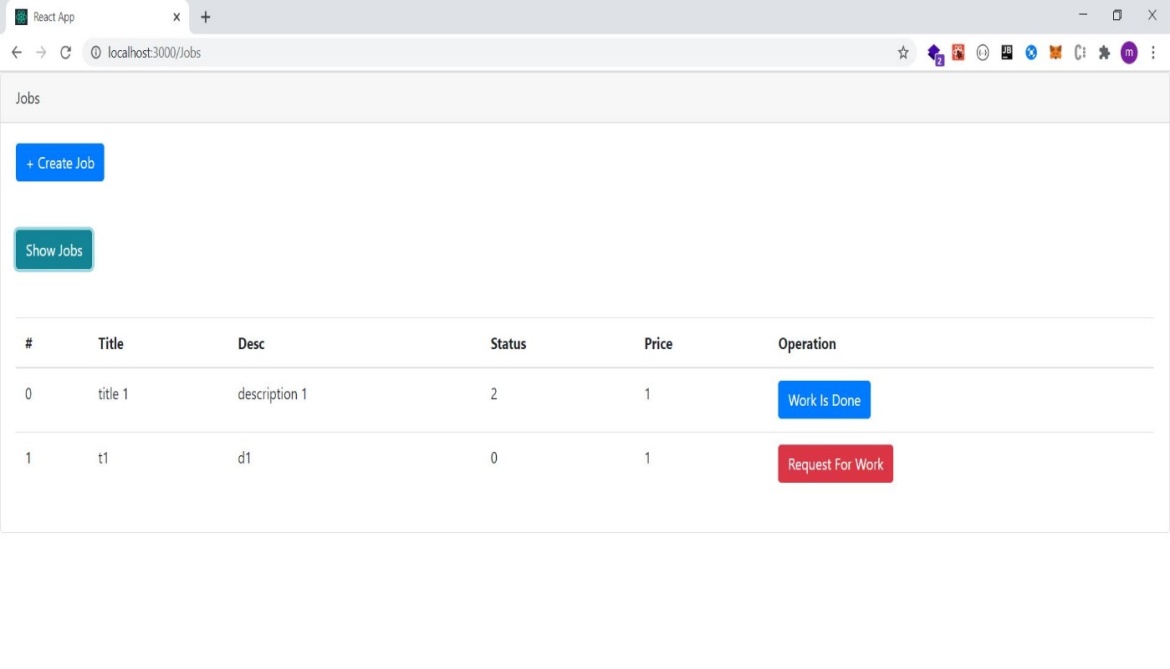


Figure39. Change status to work is done

To press the work is done button, which represents the completion of the work by the freelancer, we need to go from the first account to the second (Figure 40).

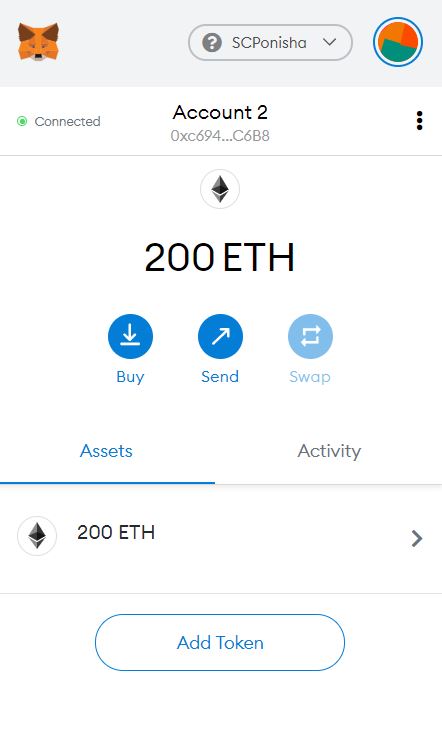


Figure40. Change account to say work is done as a freelancer

By confirming the transaction related to the completion of work by the freelancer user (Figure 41) and by pressing the work is done button, the job status is transferred from 2 to 3 (Figure 42).

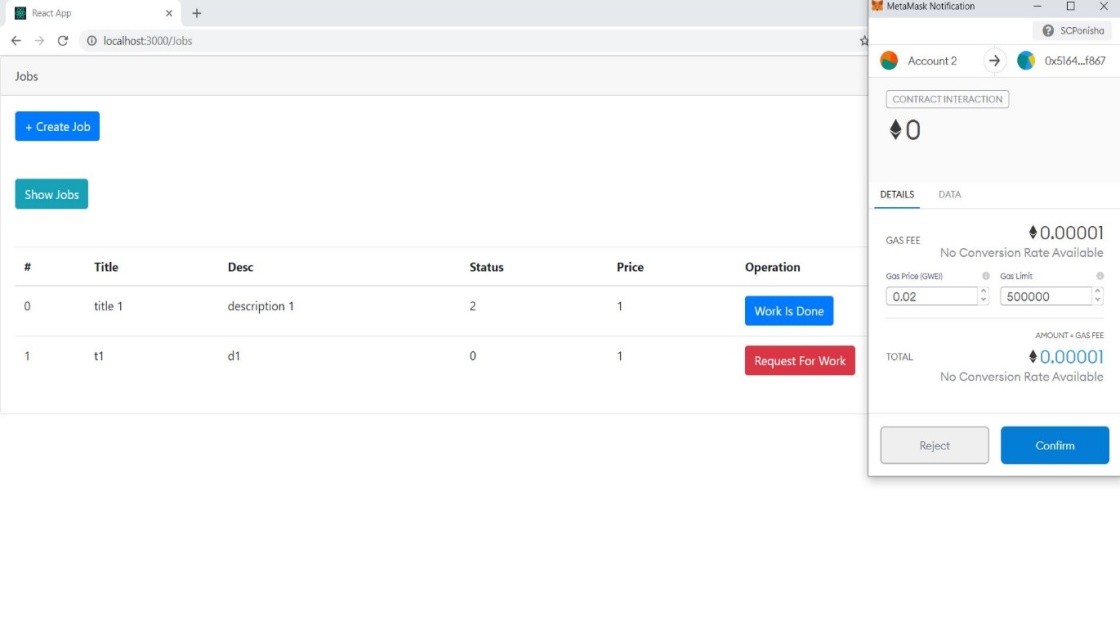


Figure41. Work is done transaction

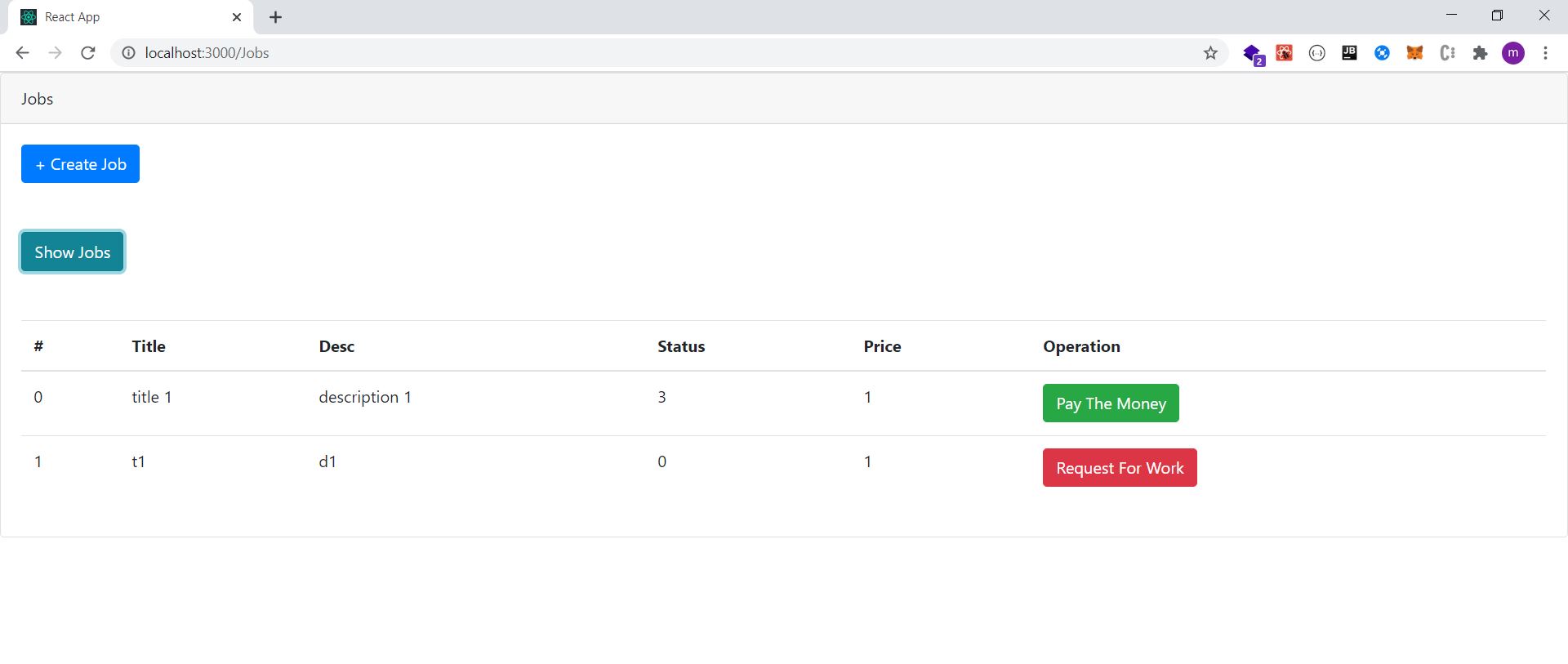


Figure42. Change status to pay the money

To pay for the project by the employer, we need to go from the account of the freelancer to the account of the employer (Figure 43).

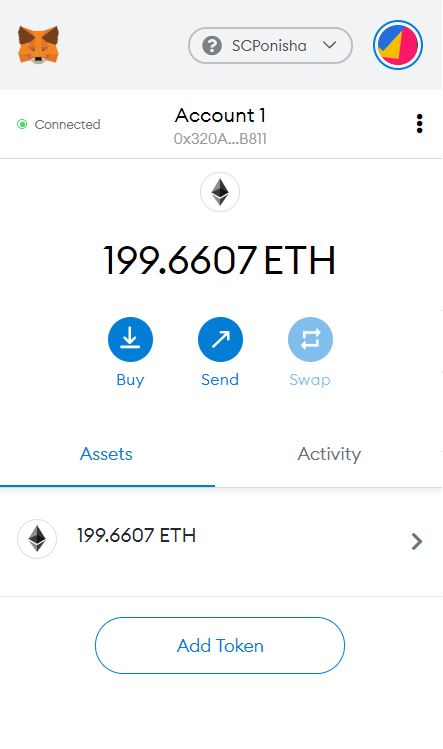


Figure43. Change account to pay the money as employer

By pressing the pay the money button, the cost related to the desired job should be transferred to the relevant freelancer. As you can see, 1 ether from account one is transferred to the smart contract and by a function, this amount is transferred to the freelancer related to this job ( Figure 44).

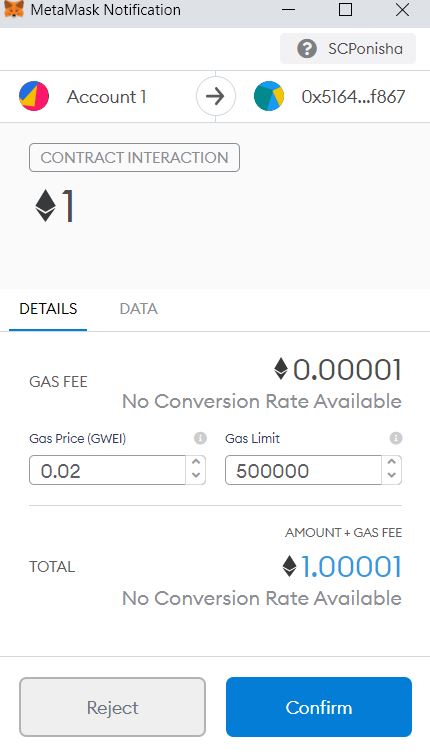
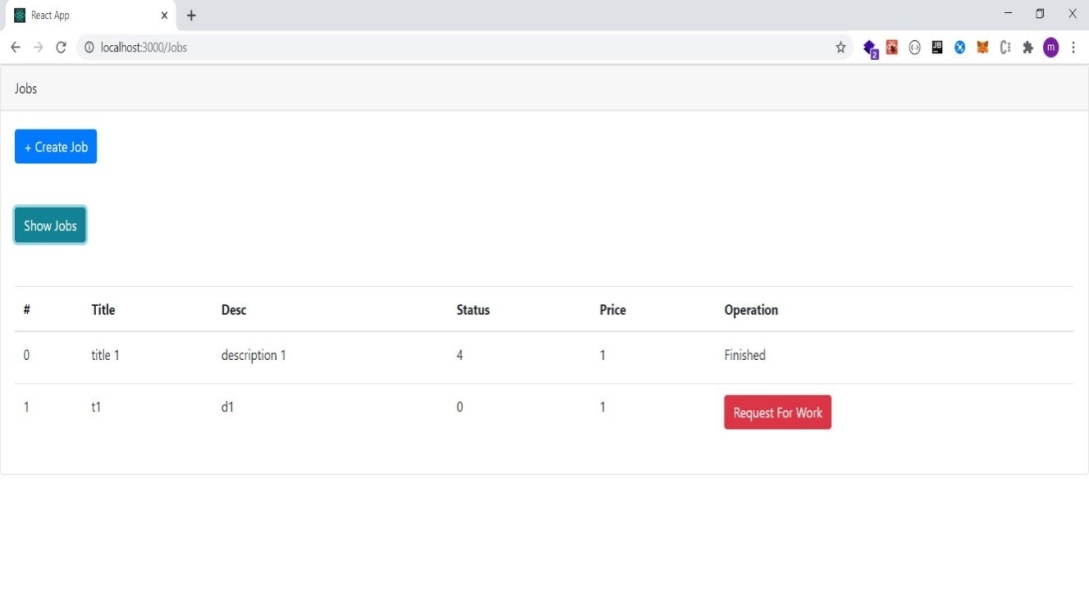


Figure44. Pay the price of job transaction

By the confirmation of the transaction related to the transfer of expenses by the employer, the job status is changed from 3 to 4 and the job is completed.



**Figure45. Change status to finished**

* **Interact with smart contracts using web3.js**

The web3.js library is used for user interaction and smart contract. Figure 46 shows the basic settings for this library.

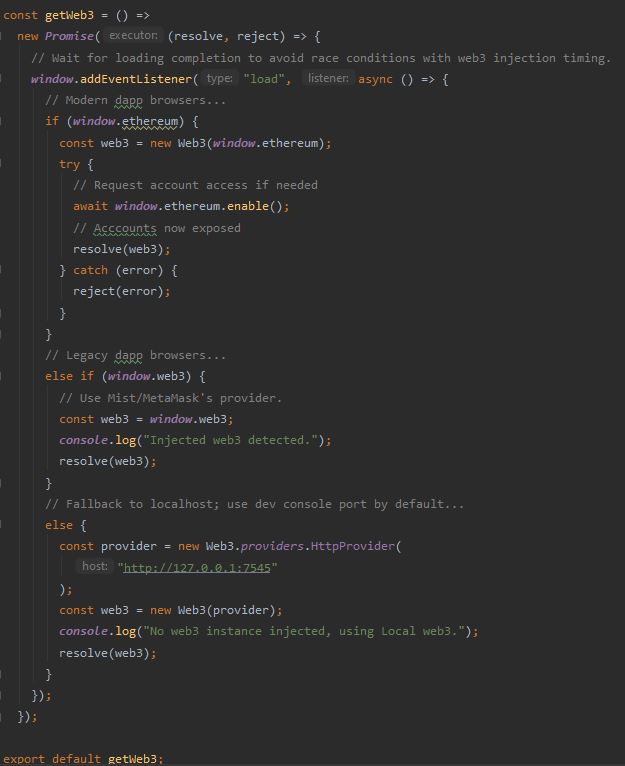
****

Figure46. using web3

After I migrating the project in truffle, you need to apply the initial settings. They are setting of accounts (Figure 47), the settings for the smart contract using the address (Figure 48) and the settings for Web3 (Figure 49).

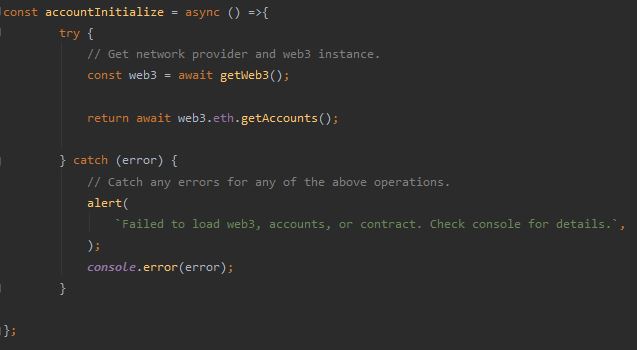
****

Figure47. account initialize

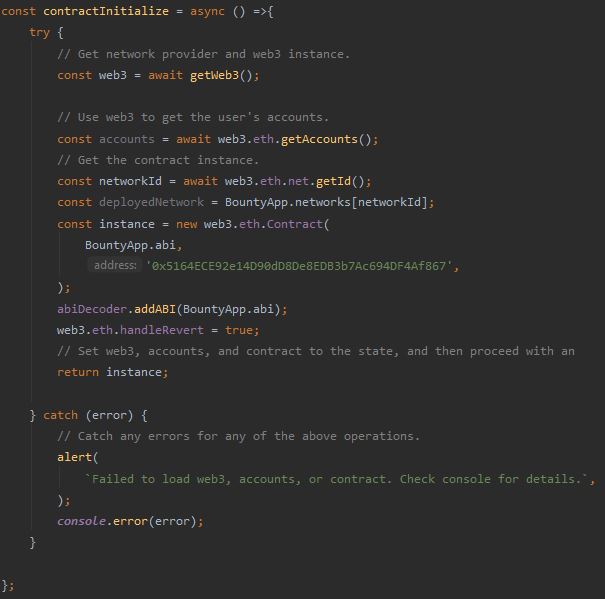
****

Figure48. contract initialize

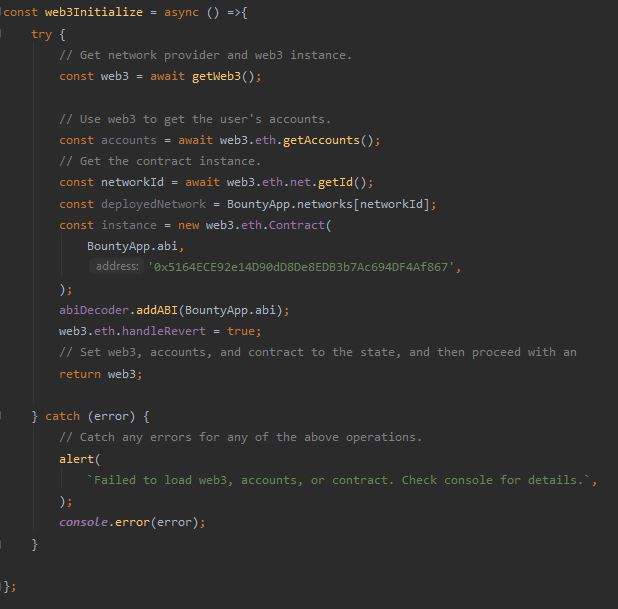
****

Figure49. Web3 initialize

The first function of the smart contract is signUp, which is called in Figure 50. The event related to this function has also been called.

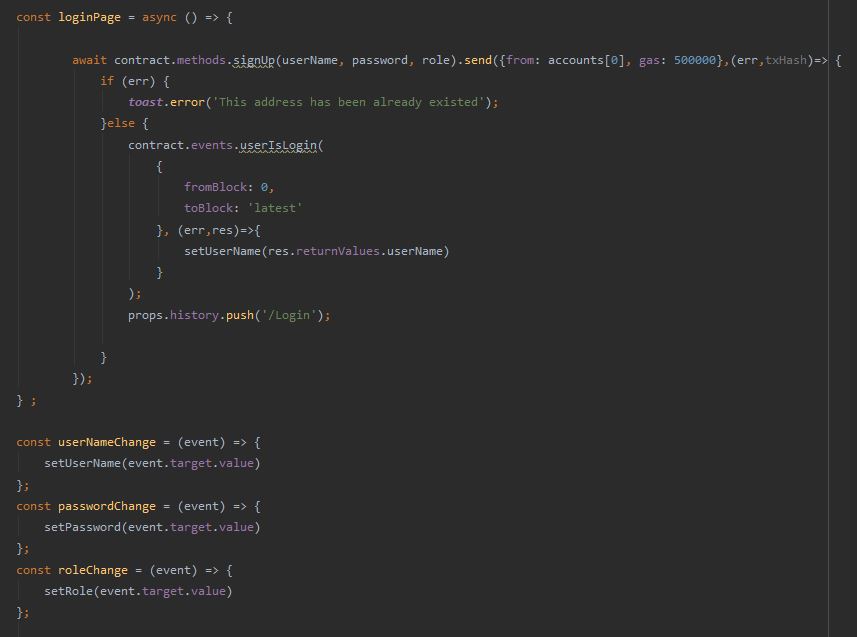
****

Figure50. Sign up method

The second function is “ login”, and related event is also called (Figure 51).

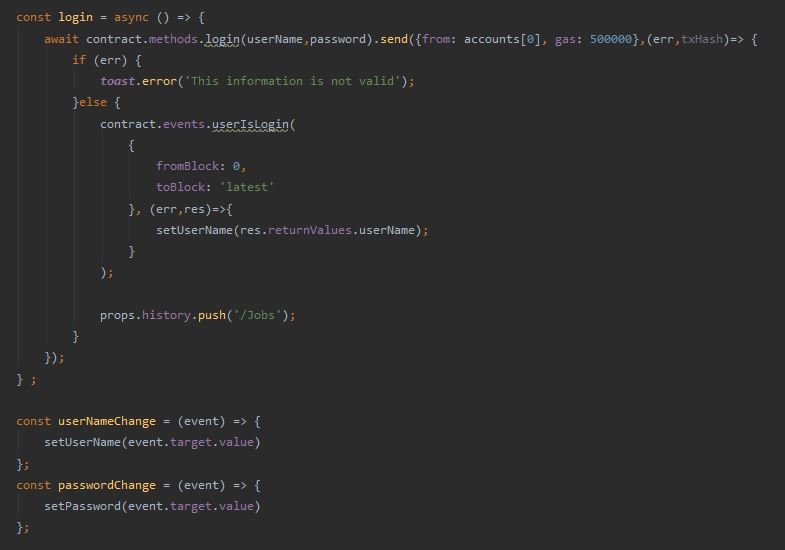
****

Figure51. Login method

The third function is related to getJobId and getJob, all of them are called according to the number of jobs (Figure 52).

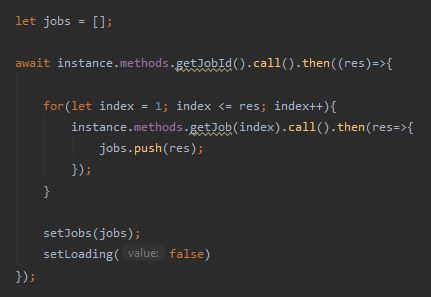
****

Figure52. Get all jobs

The next function is the freelancer requesting for the job (Figure 53).

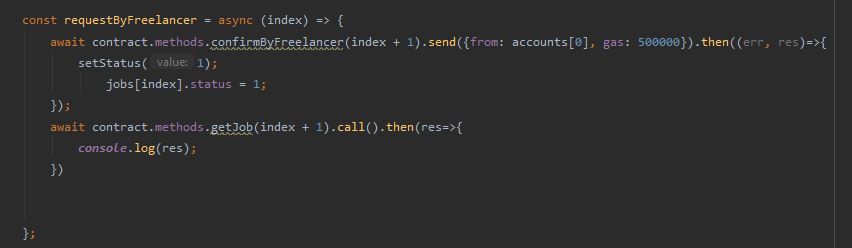
****

Figure53. Request by freelancer method

The next function is related to the approval of the freelancer by the employer for the relevant job (Figure 54).

****

Figure54. Accept by employer method

Then it is the turn of the freelancer to request the completion of the work for the relevant job (Figure 55).

****

Figure55. Submit works by freelancer method

Other functions include creating a job by the user and calling the related event (Figure 57).

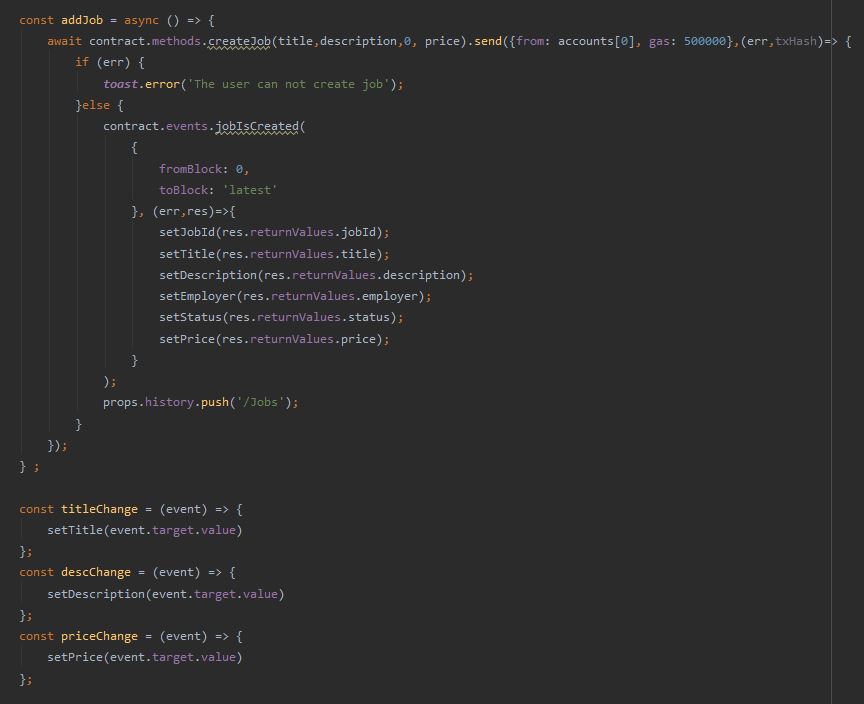
****

Figure57. Create job method

* **library**

In this project, a library has been used that is linked to the smart contract. This library is used to convert job price from ether to wei.

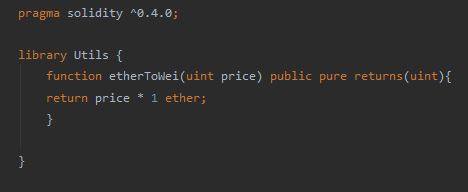


Figure57. utils library

* **tests for smart contract:**

Seven tests have been written for the smart contract which will be explained .To test, we first call the smart contract in Figure 58.



Figure58. Import contract

Then we have to make the initial settings for the tests (Figure 59).

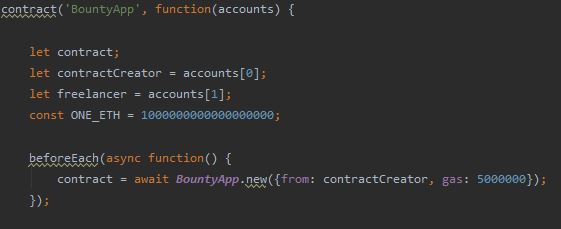


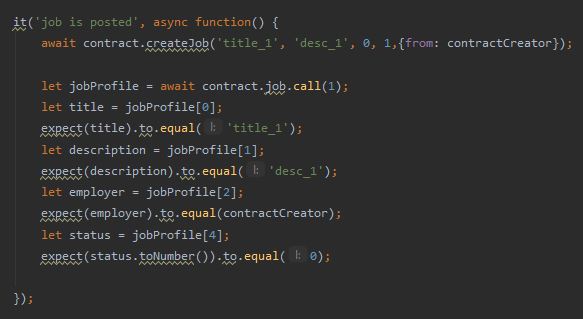
Figure59. Initialize contract test

The first test is related to user registration, in which the username, password and role of the user are compared with the desired values (Figure 60).



Figure60. User registration test

The second test is to create a job in which the title, description, status and amount of input are measured with the desired values (Figure 61).



**Figure61. create job test**

The third test is related to the freelancer request for the job, in which the input parameters are tested with the required parameters (Figure 62).



**Figure62. Request by freelancer test**

The fourth test is related to the approval of the freelancer by the employer for the job, where the input parameters are controlled with the required parameters and the test is accepted (Figure 63).



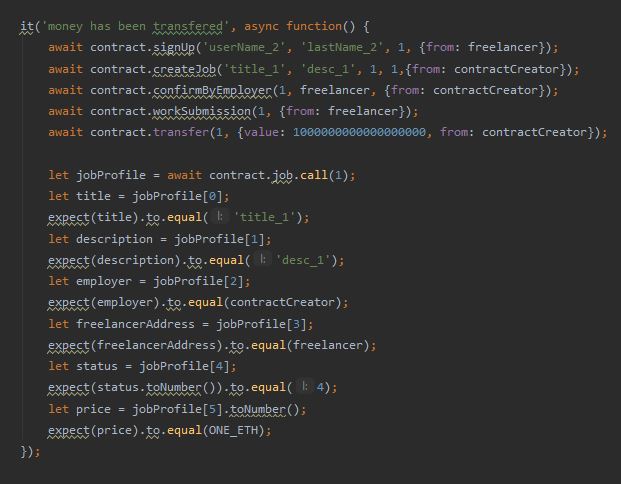
**Figure63. Confirm by employer test**

The fifth test is for the job completion request for which the input parameters are matched with the required parameters (Figure 64).



**Figure64. Job is submitted by freelancer test**

The sixth test is the cost transfer by the employer for the job, which is done and tested by determining the required amounts (Figure 65).



**Figure65. pay the money test**

The last test for the event is called jobIsFinished, the value of which is tested according to the input parameters (Figure 66).



**Figure66. Event is emitted test**

All the mentioned tests are performed using the truffle test command code in the project folder and the result can be seen in Figure 67.



**Figure67. Truffle test**