SkillSwap documentation

**Home page:**

1 - Navbar bar(header menu)

- skillswap LOGO(different logo for mobile) redirects to home pagewhen clicked

- SEARCH BAR to look for services available (with right keyword you can find services if the keyword matched with what keyword seller has put in their profile).

- CART redirects to order page

- CHAT redirects to chat page

- MY PROFILE or CREATE ACCOUNT redirects to my profile page if account is created with metamask wallet if not then create account

- CONNECT with metamask button if account is connected it reads "Connected" if user has account in skillswap it reads wallet address for example '0x42...424'

2 – Hero section(front section of home page)

-on the LEFT is the the head ‘Decentralized Freelancing BETA’ below that is text describing platform (text is in lighter gray color) below that there are two buttons ‘View Gigs’ and ‘View Contractors’

-VIEW GIGS redirects you to page where all gigs are available

-VIEW CONTRACTORS redirects to page where profile of all contracts are available

-on the RIGHT we have The Zaba skillswap logo(purple in color) and colorful blocks below that

-the BACKGROUND is in #151517 color with four blueish blob floating

3 – Category Section

-HEAD is in a box with #2f2f2f borders, in the box is a head text and a descriptive text

-all CATEGORIES – each category is in a box (similar to the box for head) which contains category name, category description, a ‘See more’ text and a logo

-each category redirects to the gig page(this page is different from above mentioned view gigs page) for their respective gig category

-BACKGROUND of this section is in #151517 color

-there are total 18 categories

4 – Seller and ETH count section

-seller count shows number of sellers in the platform

-eth count shows how much eth is available in the smart contract(it may go up and down depending on ordering and delivering going on in the platform)

-seller count is in the left and eth count in right

-numbers are over text

-there’s same background color as above sections

-this section is separated by thin border border from top and below

5 – About section

-above everything is a head in a box with #5fd3f3 color title and text about the website

-then we have three boxes which suggesting why skillswap

-same box as the head

-in the BACKGROUND we have #151517 color with same blobs same as above hero section

6 – Footer section

-LEFT skillswap copyright text and terms of services redirects to the TOS page

-RIGHT discord logo redirects the the zabas discord server

-above these is a thin border

**Gigs Page**

1 – At the top of the page is head text and below that some descriptive text.

2 – After head comes buttons with categories names on click they will take you to that category on the page.

3 – Next is the container which contains head name of the category and description of the category – then comes all the gigs which falls in that category.

4 – A gig container is a box which container picture of the gig, name of the seller, title of the gig, and fixed price. When click on this box it will redirect to the sellers profile page.

**Seller Profile Page**

1 – on LEFT we have seller information which includes

-profile pic

-Name and icon to share his/her profile

-rating

-one line bio

-button to contact seller (redirects to the chat)

-website link

-twitter handle

-country

-language he/she speaks

-About the seller

-skills seller has

-education/qualifications

2 – on the RIGHT we have seller’s gig information which includes

-gig title and gig picture below that

-service description

-Prices of the service

-portfolio images

-Review section contains rating(stars) seller has got, address of the buyer and feedback for seller

**Contractors Page**

1. Head same as Gigs page contains head text and descriptive text.
2. Below head comes profile of all contractors, it is a box which includes profile picture of the contractor, name, one line bio. On click redirects to the contractor profile page.

**Contractor Profile Page**

1. Everything is in the middle includes
2. Profile picture of contractor
3. Website link, twitter handle, location, email and phone number in one place beside pofile pic
4. Name of the contractor
5. One line bio of the contractor
6. About the contractor

**My Profile Page**

1. Same as Seller profile page if the account is made as a freelancer, except this page contains an edit button on click seller can edit their profile. This page do not have contact seller button as Seller profile page
2. Same as Contractor Profile page if the account is made as a contractor except this page also contains an edit button on click se contractors can edit their profile.

**Chat Page**

1. Left side is a container which contains address, name and profile pic of everyone user can chat with, it is scrollable
2. In the Middle we have name and address user is chatting with currently at the top. Then comes all the chats, users messages are in blue color and receiver messages are in white, these also contains date and time when the chat is sent.
3. Then we have input area where users can type and a send button to send the message, it also contains offer button if the user is a seller to offer the contractors.
4. On the Right we have Offer section which shows the offer that seller has made with button to withdraw the offer if the user is a seller or accept and decline the offer if the user is a contractor.

**SkillSwap smart contract working**

**The smart contract of skillswap has these functionalities:**

1. Setting up and storing sellers and buyers profile – storing uri(metadata of both in smart contract) - function setProfile(string memory \_uri, uint8 userId) public

2. Updating profile – changing the uri(metadata of both seller and buyer in the SC) - function updateProfile(string memory \_uri, uint256 \_id, uint8 userId) public

3. Placing order – run by buyer (buyer pays the gas fees and amount as per the deal) function placeOrder(address \_seller, uint256 \_amount, uint256 \_duration) public payable

4. Confirming delivery – run by buyer (buyer pays the gas fees) function confirmDelivery(address \_seller, string memory \_uri, uint256 \_id) public

5. Transferring eth to the seller – run by seller (seller pays the gas fees) function toSeller(address \_buyer) public payable

6. Cancelling order – seller and buyer both can run this function function cancelOrder(address \_seller, address \_buyer) public

7. Deadline met - function deadlineMet(address \_seller, address \_buyer) public

8. Withdraw eth by the owner – only owner of the smart contract the run function withDrawbalance() public

**Working**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.9;

* License and solidity verse used

address owner;

constructor(){owner = msg.sender;}

* Declaring whoever DEPLOYS the contract is the OWNER

enum escrow{offered, ordered, delivered, complete}

* Declaring system, offered means transaction is at 0, ordered at 1, delivered at 2, and complete at 3.
* Everything runs in order of escrow, for example if a function is at ordered it cannot jump to complete. EXCEPTION – function cancelOrder() and function deadlineMet()

mapping(address=>mapping(address=>escrow)) public Transaction;

* Making mapping of seller address => buyer address => escrow{offered, ordered, delivered, complete} and naming this as Transaction
* When a buyer ‘A’ places an order with a seller ‘B’ it works something like this B => A => escrow
* More on this later

struct Deal{uint256 amount; uint256 duration; address seller; address buyer; bool inProgress; }

* Here we’re declaring a list of all the things that will be involved in a deal between buyer and seller.
* Amount is the price that the seller is charging for his/her work.
* Duration is time limit before the deal ends.
* Seller is the address of the seller.
* Buyer is the address of the buyer.
* inProgress is a bool(true/false) if the order is placed it is true means in progress and if the order is complete or not placed yet it is false means NOT in progress

mapping(address=>mapping(address=>Deal)) public deal;

* Making a mapping of seller => buyer => Deal and naming it as deal, this is similar to the Transaction

function placeOrder(address \_seller, uint256 \_amount, uint256 \_duration) public payable{

require(isSeller[\_seller], "not a seller");

require(isBuyer[msg.sender], "not a buyer");

require(deal[\_seller][msg.sender].inProgress == false, "in process");

require(\_seller != msg.sender, "Cannot order yourself");

require(msg.value == \_amount+\_amount\*1/10, "set the right amount");

deal[\_seller][msg.sender] = Deal(\_amount, \_duration + block.timestamp, \_seller, msg.sender, true);

Transaction[\_seller][msg.sender] = escrow.ordered;

}

* A function ‘placeOrder()’, it takes three parameters ‘\_seller’, ‘\_amount’, ‘\_duration’
* ‘\_seller’ is the address of the seller
* ‘\_amount’ is the price seller is charging for his/her work
* ‘\_duration’ is the time limit
* require statements explained – 1st .function will run only if ‘\_seller’ is the address of a seller suppose if the ‘\_seller’ is not a sellers address or the seller has not made an account in skillswap as a seller this function will not be run.

2nd . function will run only if the buyer has paid gas fees. 3rd . function will run only if deal between seller and buyer is not in progress means buyer cannot place order again if the order is still in progress between seller and buyer.

4th . function will not run if the seller is running the function in other words paying the gas fees. 5th. Function will only run if the buyer is paying the ‘\_amount’ + 10% of ‘\_amount’.

* What this function does is make a object of \_amount, \_duration + current time in unix timestamp, \_seller, buyer address, and true which is in progress and then storing this object in the mapping of buyer and seller
* Now here we make the Transaction between seller and buyer ‘ordered’ which is escrow at 1.

    function confirmDelivery(address \_seller, string memory \_uri, uint256 \_id) public {

        require(Transaction[\_seller][msg.sender] == escrow.ordered, "order not yet placed");

        deal[\_seller][msg.sender].duration = block.timestamp;

        sellerProfile[\_id].uri = \_uri;

        Transaction[\_seller][msg.sender] = escrow.delivered;

    }

* Now after seller delivers the work buyer can confirm delivery and proceed with escrow
* confirmDelivery takes three parameters ‘\_seller’, ‘\_uri’, ‘\_id’
* require statement allows to run the function only if a specific condition is met
* here function will run only if the Transaction between seller and buyer is ordered means escrow is at ordered (1)
* in other words this function will run only after placeOrder() func. and before toSeller() func.
* Now what this function does is it stops the count down by making deadline equal to current date and time.
* Then it updates the uri of the seller to add reviews taken by the buyer
* Now finally it proceeds with escrow by making Transaction between seller and buyer = delivered.
* This means now escrow is at delivered (2)

    function toSeller(address \_buyer) public payable{

        require(Transaction[msg.sender][\_buyer] == escrow.delivered, "order not yet delivered");

        totalCommision += deal[msg.sender][\_buyer].amount \* 1/10;

        payable(msg.sender).transfer(deal[msg.sender][\_buyer].amount);

        Transaction[msg.sender][\_buyer] = escrow.complete;

        delete deal[msg.sender][\_buyer];

    }

* After the seller has delivered the work and buyer confirms the delivery by running the confirmDelivery() func. we can now proceed with escrow and toSeller() func.
* This is run by seller only in other words seller pays the gas fees. It takes one parameter ‘\_buyer’ address which will later be used in the function.
* Like before require statement check if the escrow is at delivered or not, if not the function will not run.
* First this function adds commission in totalCommision variable which later will be used to withdraw commission from the SC by the owner
* Now we transfer the amount to the seller
* Finally we mark escrow as complete
* Now we delete the deal between seller and buyer so that they could make deal again in the future

    function cancelOrder(address \_seller, address \_buyer) public {

        require(Transaction[\_seller][\_buyer] == escrow.ordered, "order already in process");

        payable(\_buyer).transfer(deal[\_seller][\_buyer].amount);

        Transaction[\_seller][\_buyer] = escrow.complete;

        delete deal[\_seller][\_buyer];

    }

* This function is used to cancel order as the name suggests cancelOrder(…)
* The function takes two parameters ‘\_seller’ address and ‘\_buyer’ address.
* It can be called by both seller and buyer.
* Before running it check (require statement again) if the seller and buyer have made an order or not, which means it checks if the escrow is at ordered. Suppose if the escrow is at delivered which means the work is already delivered by the seller to buyer so there’s no point of cancelling the order.
* Now it transfers the eth back to the buyer Full Amount.
* Next it marks escrow as complete.
* Then it deletes the deal between seller and buyer so that they could make deal again in the future.

    function deadlineMet(address \_seller, address \_buyer) public {

        require(Transaction[\_seller][\_buyer] == escrow.ordered, "order already in process");

        require(deal[\_seller][\_buyer].duration<block.timestamp, "deadline not met yet!");

        payable(\_buyer).transfer(deal[\_seller][\_buyer].amount);

        Transaction[\_seller][\_buyer] = escrow.complete;

        delete deal[\_seller][\_buyer];

    }

* This function runs when the deadline is met
* It takes two parameters \_seller address and \_buyer address.
* Before running it check (require statement again) if the seller and buyer have made an order or not, which means it checks if the escrow is at ordered. Suppose if the escrow is at delivered which means the work is already delivered by the seller to buyer so there’s no point of running this function.
* Now it checks it the deadline is really met or not by comparing the current timestamp with the timestamp mentioned while making order
* Now it transfers the eth back to the buyer Full Amount.
* Next it marks escrow as complete.
* Then it deletes the deal between seller and buyer so that they could make deal again in the future.

-----------------------HERE ALL THE TRANSACTION AND ESCROW PART ENDS-------------------

    modifier onlyOwner{

        require(msg.sender == owner, "Not the owner");

        \_;

    }

* This is modifier onlyOwner, whichever function has ‘onlyOwner’ after its name means only owner of the smart contract can run that function
* function withDrawbalance() public onlyOwner{
* payable(owner).transfer(totalCommision);
* totalCommision=0;
* }
* This function is used to withdraw eth from the contract
* Can only be run by the owner of the SC
* Not every eth can be transferred to the owner only the commission which gets from a deal between seller and buyer.
* Suppose there’s a deal between two person, buyer offers 1ETH to the seller. But buyer gives 1.1Eth to the SC, extra eth as commission. Now when the escrow is complete and seller gets his/her share, what is left is added to the totalCommission variable which is here 0.1 ETH. Now the owner can transfer this 0.1Eth to his/her wallet.
* Next we make totalCommission to 0 again.

    function removeUser(uint8 userId,uint256 \_id, address \_account) public onlyOwner{

        if(userId == 0){

            require(isSeller[\_account], "not a seller");

            delete sellerProfile[\_id];

            isSeller[\_account] = false;

            --noOfSellers;

        }

        if(userId ==1){

            require(isBuyer[\_account], "not a buyer");

            delete buyerProfile[\_id];

            isBuyer[\_account] = false;

            --noOfBuyers;

        }

    }

* This function is simply used to remove user
* Only owner can run this function

---------------------SETTING UP PROFILE AND UPDATING IT------------------------------------

    struct Profile{

        address account;

        uint256 id;

        string uri;

    }

* The variable Profile makes an object of account address, id, and uri(metadata)

    uint256 public noOfSellers;

    uint256 public noOfBuyers;

* Simple counter of number of seller and number of buyer

    mapping(uint256=>Profile) public sellerProfile;

    mapping(address=>bool) public isSeller;

* Mapping contains seller profile attached with a number which is id of the seller
* 2nd Line is a mapping of address and bool, means address of the seller is registered as a seller in the platform or not.

    mapping(uint256=>Profile) public buyerProfile;

    mapping(address=>bool) public isBuyer;

* This is the exact same thing as above but for buyers

    function setProfile(string memory \_uri, uint8 userId) public {

        require(isSeller[msg.sender]==false, "Already a seller");

        require(isBuyer[msg.sender]==false, "Already a buyer");

        require(userId <= 1, "invalid choice");

        if(userId == 0){

            ++noOfSellers;

            sellerProfile[noOfSellers] = Profile(msg.sender, noOfSellers, \_uri);

            isSeller[msg.sender]=true;

        }else if(userId == 1) {

            ++noOfBuyers;

            buyerProfile[noOfBuyers] = Profile(msg.sender, noOfBuyers, \_uri);

            isBuyer[msg.sender]=true;

        }

    }

* This function is used to store uri of the buyer or seller in the contract
* Takes two parameters \_uri and userId, note that userId is not the id of seller and buyer
* userId 0 means user is make account as a seller and 1 means as a buyer
* it checks if the user is already a seller or buyer if so the function will revert(not run).
* Next it checks if the userId is less than equal 1 or not, userId can only be 0 or 1, 0 means seller and 1 means buyer
* Next it increases the total number of seller or buyer as per userId.
* Then it stores the address of seller or buyer, number of seller or buyer (which is id), and \_uri (metadata) in variable Profile which is attached to id of seller or buyer(no of seller or buyer)
* Then it marks isSeller or isBuyer as true attached to their address respectively, means seller has made an account or buyer has made an account in the platform.

    function updateProfile(string memory \_uri, uint256 \_id, uint8 userId) public{

        require(userId <= 1, "invalid choice");

        if(userId == 0){

            require(isSeller[msg.sender]==true, "not a seller");

            require(sellerProfile[\_id].account == msg.sender, "NOT your profile");

             sellerProfile[\_id].uri = \_uri;

        }else if(userId == 1){

            require(isBuyer[msg.sender]==true, "not a buyer");

            require(buyerProfile[\_id].account == msg.sender, "NOT your profile");

            buyerProfile[\_id].uri = \_uri;

        }

    }

* This function updates the profile of seller or buyer as per the given parameter userId(0 or 1)
* First it checks that userId must be less than equal to 1 which is 0 or 1
* Now it checks if the seller or the buyer has already made an account or not, if not it reverts
* Next it checks if the seller or the buyer are updating their owner profile, some else profile cannot be updated here.
* Then it changes their uri (metadata) with a new uri.

--------------------------------------------------------------------------------------------------------------------------------------

*This is the documentation I got from chatGPT and it tells everything about the smart contract maybe better than me. I’m leaving it here in case someone doesn’t get what I’ve written.*

Smart Contract Documentation: SkillSwap

This smart contract is designed to facilitate the exchange of skills or services between buyers and sellers on the Ethereum blockchain. The contract is implemented using Solidity programming language and can be deployed on the Ethereum Virtual Machine (EVM).

**Contract Details**

* + Name: SkillSwap
  + License: MIT
  + Solidity Version: 0.8.9

**Contract Variables**

1. owner: Address variable to store the owner's address
2. enum escrow{offered, ordered, delivered, complete}: Enum variable to keep track of the escrow status of a transaction
3. mapping(address=>mapping(address=>escrow)) public Transaction;: Nested mapping to store transaction details of each buyer and seller
4. struct Deal{...}: Struct to store deal details such as the amount, duration, seller, buyer, and progress status of each deal
5. struct Profile{...}: Struct to store profile details such as the account address, id, and uri of each buyer and seller
6. uint256 public noOfSellers: Counter variable to keep track of the number of sellers registered on the platform
7. uint256 public noOfBuyers: Counter variable to keep track of the number of buyers registered on the platform
8. mapping(uint256=>Profile) public sellerProfile: Mapping to store seller profile details using their id as a key
9. mapping(address=>bool) public isSeller: Mapping to store the status of whether an account is a registered seller or not
10. mapping(uint256=>Profile) public buyerProfile: Mapping to store buyer profile details using their id as a key
11. mapping(address=>bool) public isBuyer: Mapping to store the status of whether an account is a registered buyer or not
12. mapping(address=>mapping(address=>Deal)) public deal: Nested mapping to store the details of each deal between a seller and a buyer
13. uint256 public totalCommision: Variable to keep track of the total commission earned by the platform

**Contract Functions**

1. constructor(): Constructor function to set the owner's address
2. setProfile(...): Function to set the profile of a buyer or a seller
3. updateProfile(...): Function to update the profile of a buyer or a seller
4. placeOrder(...): Function to place an order for a service or a skill
5. confirmDelivery(...): Function to confirm the delivery of a service or a skill
6. toSeller(...): Function to transfer the payment to a seller after the delivery of the service or the skill
7. cancelOrder(...): Function to cancel an order placed by a buyer
8. deadlineMet(...): Function to cancel an order when the deadline is not met by the seller
9. withDrawbalance(): Function to withdraw the total commission earned by the platform
10. removeUser(...): Function to remove a buyer or a seller from the platform

**Modifiers**

1. onlyOwner: Modifier to restrict access to certain functions to the owner of the contract.

**Note**

This documentation is provided for informational purposes only and is not intended as investment advice, legal advice, or any other form of advice. The use of this contract is entirely at your own risk.