

Blockchain and cryptocurrencies

Blockchain-based game

Keran Jegasothy
Qazim Muçodema
Jason Shuyinta

Contents

1	Introduction	3
2	Design	4
2.1	Game features	6
2.1.1	Create an account	6
2.1.2	Access your profile	7
2.1.3	Catch new monsters	7
2.1.4	Fight against other players	7
2.1.5	Access the shop	7
2.2	Admin features	8
3	Implementation	9
3.1	Tech stack	9
3.2	Solidity	9
3.2.1	Account	10
3.2.2	Monsters	10
3.2.3	Numeri random	11
3.2.4	Cattura	12
3.2.5	Combatti	13
3.2.6	Shop	15
3.2.7	Admin-executable operations	16
3.3	Web3 Javascript	17
4	Screens	18
4.1	Homepage	18
4.2	Profile	18
4.3	Play	19

4.3.1	Catch	19
4.3.2	Fight	19
4.4	Shop	20
4.4.1	Snipping Tools	20
4.4.2	Sell	21
4.4.3	Upgrade	21
4.4.4	Marketplace	22
4.5	Owner	22
4.5.1	Add new monster	22
4.5.2	Add new move	23
4.5.3	Withdraw	23

1. Introduction

We decided to build an interactive game using "Pokemon" and "Axie Infinity" as a source of inspiration. The players will be able to own a collection of monsters and exchange them with each other within the shop. At the moment an online fighting system between the various players has not yet been implemented, but offline battles can be carried out where attacks are randomly chosen. The interactions between and on the monsters are being implemented using smart contracts. In this documentation we will focus on the explanation of the most relevant elements of the game, therefore we will concentrate on the effects of the smart contracts. Other aspects of the implementation of the game are feasible and at times will only be mentioned.

2. Design

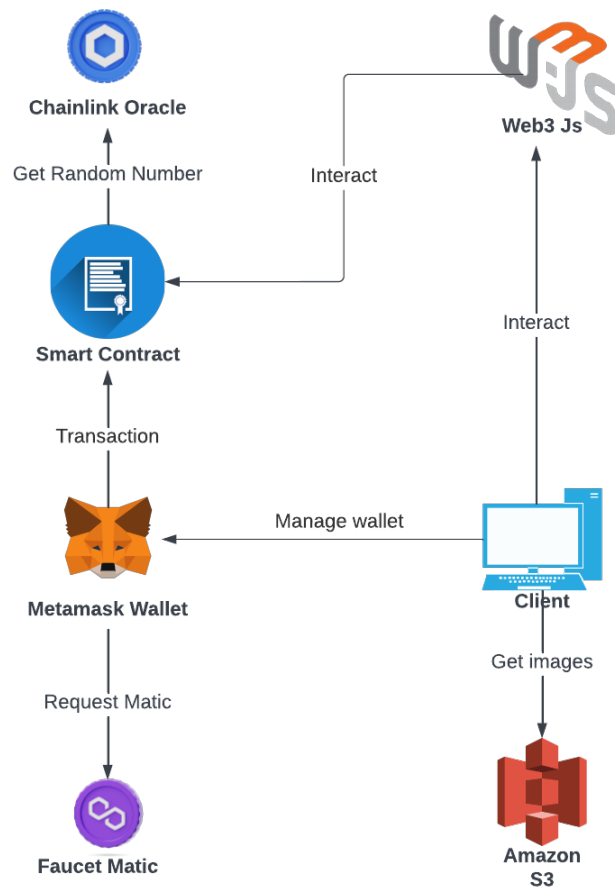


Figure 2.1: Architecture diagram - Client

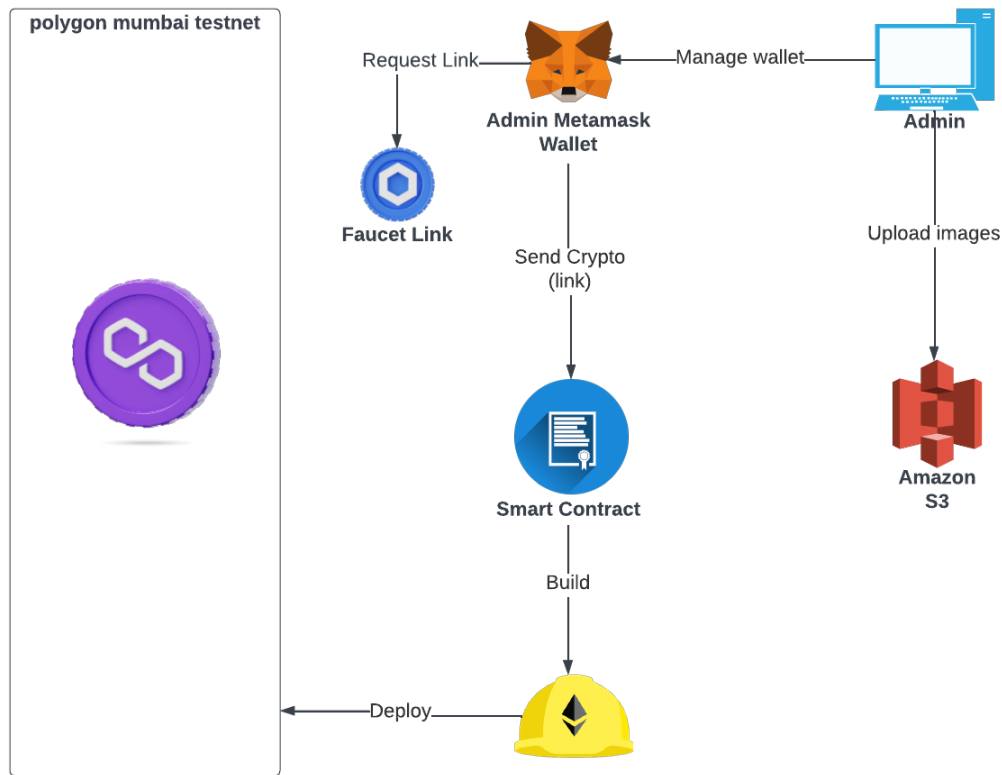


Figure 2.2: Architecture diagram - Admin

To describe our architecture let's consider the diagram above. Starting off with the Smart Contract, as it represents our application. The build function is necessary as our first step in order to deploy the main contract inside of the Polygon Mumbai Testnet. Another fundamental step is sending Link tokens to our smart contract so it will be possible to obtain an off-chain random number from Chainlink which we will utilize during the search for a wild monster or during an attempt to catch it. Obtaining a random number directly from the blockchain is not considered a "secure" approach, as we are living in a deterministic environment, therefore we would have the same seed to produce pseudo-random elements. In the case that the expected result is known or identifiable we could incur in an application exploit. To avoid this possibility we decided to opt for an external source to obtain our random number, using the Chainlink oracle. In the moment the admin deploys the contract, the application is reachable from every client. Keep in mind that the admin is itself a client, so from his account he could operate at the same level of a normal client. The majority of the operations possible in the game

(i.e. catch a wild monster, buy from the shop, find new monsters...), are all operations made available from the smart-contract that operates through Link cryptocurrency and Matic. For this reason every client need a wallet from which he will make transactions in order to play and activate the smart contract functions. The wallet gets populated with fake-crypto using platforms such as Faucet Matic and Faucet Link. As for the monster, we have chosen to opt for AWS S3 buckets as our storage options. The link to every monster images gets saved inside of the smart contract e afterwards the user will visualize it on the web page.

2.1 Game features

Each player has the possibility to play and enrich his collections of monsters in order to be more competitive. In the next paragraph we will list the main features of the game.

- Create an account
- Access your profile
- Catch new monsters
- Fight against other players
- Access the shop

Here we have an extract of the navbar that allows to access the various sections.



Figure 2.3: Navbar

2.1.1 Create an account

A user in order to create an account needs to have a Metamask wallet (to every Metamask account can be associated a single gaming account). It is necessary that the user connects its wallet with the web page. Afterwards there will be the possibility to insert a username. After creating the account, the user will have the possibility to choose his first monster:

- **Firemonster:** monster of type fire at level 1.
- **Watermonster:** monster of type water at level 1.
- **Grassmonster:** monster of type grass at level 1.

2.1.2 Access your profile

Access the section *Profile*, the user has the possibility to visualize its data:

- username
- owned Matic
- player level
- number of victories\number of losses
- owned monsters
- number of owned catching tools

Plus, clicking on an owned monster, the player has the possibility to make it its currently active monster.

2.1.3 Catch new monsters

Accessing the *play* and then the *catch* section, it will be shown to the user a random monster which can be caught. If the found monster is not interesting to the player it could be skipped by selecting the button. If the monster is, on the other hand, interesting, the player will have the possibility to try to catch it using one of the catching tools (see section 1.5). Until the monster is caught the player will have the possibility to use all of its catching tools.

2.1.4 Fight against other players

Accessing the *play* and then the *fight* section, it will be shown to the user a random player to fight against. For the battle, the active monster of both of the players will be user. The user will be able to select an attack to use between the ones he has available on the active monster. While the opponents monster attack will be chosen randomly. Afterwards a battle will take place, and the monsters life points will be compared and the one who will have more is the winner of the battle.

2.1.5 Access the shop

Access the *Shop*, the user has the possibility to:

- buy catching tools:
 - **Normal tool:** costs 0.005 Matic and it has a catching chance of 30%

- **Mega tool:** costa 0.006 Matic and it has a catching chance of 60%
- **Ultra tool:** costa 0.007 mMatic and it has a catching chance of 90%
- sell a monster he owns
- upgrade a monster he owns by increasing its level or adding new attacks to it
- buy monsters sold by other players

2.2 Admin features

The admin is he who deploys the smart-contract inside of the Polygon blockchain. At the moment the application allows for only a single user to be an admin. In addition it is configured as responsible for any update of the platform, as it is the only one able to add new monsters or moves for the players to user. When operations are made inside of the shop, the admin is able to withdraw the Matic that players paid for new attacks o catching tools. Following are the list of operations we just described:

- insert a new monster.
- insert a new buyable attack.
- visualize how many Matics are left on the contract and withdraw them.

3. Implementation

The project has a Client side in React in which the users, as said before, will be able to interact for example by fighting other players. We are not going to describe this side as it is outside of the project scope. For this reason we will concentrate on the functions developed in Solidity and the React configuration for calling these same functions.

3.1 Tech stack

The tech stack used in the following project is the following:

- **Polygon Mumbai Testnet**
- **Solidity** for the creation of smart contracts
- **Hardhat** to build and deploy smart contracts on the blockchain.
- **Metamask** as our preferred crypto-wallet.
- **Chainlink** to obtain a random number off-chain.
- **Ganache** to create a fake-blockchain for testing purposes.
- **Amazon S3** to save images.
- **Web3 Js** allows the client to interact with the smart contracts
- **Faucet Matic** and **Faucet Link** to add fake Matic and fake Link to the contract
- **HTML, CSS, Javascript** and **React** for the User Interface.

3.2 Solidity

In the following chapter we will report the features implemented in Solidity. Solidity is an object-oriented programming language for the implementation of smart contracts for various blockchains. The following list shows the main functions implemented:

3.2.1 Account

```
1 // Function to create new user associate to wallet
2 function createUser(string memory username) public returns (bool) {
3     if (addressToUser[msg.sender].exist == true) {
4         return false;
5     } else {
6         addressToUser[msg.sender] = User( username,1, 0, 0, 0, msg.sender,
7         true);
8         users.push(msg.sender);
9         return true;
10    }
```

The above function allow the user to create his account. Before creating an account it is checked whether the account is already associated to a wallet. In case it isn't the account is then created.

```
1 // Function to login
2 function login() public view returns (string memory name) {
3     if (addressToUser[msg.sender].exist == true) {
4         return addressToUser[msg.sender].name;
5     } else {
6         return "";
7     }
8 }
```

The above function allows the user to login to the game.

3.2.2 Monsters

```
1 // Function to save monsters in blockchain whit they moves
2 function createDefaultMonster(string memory name, uint8 typeOfMonster,
3     uint16 healthPoints, string memory url, string memory moveName, uint16
4     damage, uint8 typeOfMove) public {
5     idToDefaultMonster[numberOfFindableMonster] = DefaultMonster(name,
6     typeOfMonster, numberOfFindableMonster, healthPoints, url);
7     findableMonster.push( DefaultMonster(name, typeOfMonster,
8     numberOfFindableMonster, healthPoints, url));
9     idToDefaultMoves[numberOfFindableMonster] = Move( moveName, damage,
10    typeOfMove);
11    numberOfFindableMonster = numberOfFindableMonster.add(1);
12 }
```

The above function allows to add new monsters inside of the blockchain. At the moment of the loading of the contract on the blockchain, this function gets called for it to insert the first findable monsters inside of the game.

```

1 // Function to assign findable monster to user
2 function addFindableMonsterToUser(uint256 monsterID) public returns (uint256) {
3     string memory nameMonster = idToDefaultMonster[monsterID].name;
4     uint8 typeOfMonster = idToDefaultMonster[monsterID].typeOfMonster;
5     uint256 GlobalId = monsterCount;
6     uint16 level = 1;
7     uint16 healthPoints = idToDefaultMonster[monsterID].healthPoints;
8     string memory urlImg = idToDefaultMonster[monsterID].urlImg;
9     addressToMonster[msg.sender].push(
10         Monster(nameMonster, typeOfMonster, GlobalId, level, healthPoints,
11             urlImg)
12     );
13     addDefaultMoveToPersonalMonster(monsterID, GlobalId);
14     allMonsters.push(
15         Monster(nameMonster, typeOfMonster, GlobalId, level, healthPoints,
16             urlImg)
17     );
18     monsterCount = monsterCount.add(1);
19     return GlobalId;
20 }

```

The above code allows to assign to the user the monster that is has captured.

```

1 // Function to change the active monster
2 function setActiveMonster(uint256 globalId) public returns (bool) {
3     bool isOwnerOfMonster = checkIsOwner(globalId);
4     bool isMonsterOnSell = checkMonsterIsOnSell(globalId);
5     if (isOwnerOfMonster == true && isMonsterOnSell == false) {
6         addressToUser[msg.sender].activeMonster = globalId;
7         return true;
8     } else {
9         return false;
10    }
11 }

```

The above function allows the user to change its active monster.

3.2.3 Numeri random

```

1 //Request Randomness
2 function getRandomNumber() public returns (bytes32 requestId) {
3     randomResult = 0;
4     require(
5         LINK.balanceOf(address(this)) >= fee,
6         "Not enough LINK - fill contract with faucet"
7     );
8     return requestRandomness(keyHash, fee);
9 }

```

```

10
11 //Callback function used by VRF Coordinator
12 function fulfillRandomness(bytes32 requestId, uint256 randomness) internal
    override {
13     randomResult = randomness;
14 }
15
16 // Getter
17 function getNum() public view returns (uint256) {
18     return randomResult;
19 }

```

The above code allows to make a request to the Chainlink oracle to obtain a random number.

3.2.4 Cattura

```

1 // Function to get a random findable monster from a random number
2 function getRandomMonster() public view returns (uint256) {
3     return randomResult % numberOfFindableMonster;
4 }

```

The above function is called the moment a user tries to capture a monster. In particular the code allows to select a catchable monster using a random number.

```

1 // Function to define if the catch have success or not
2 function captureMonster(uint256 typeOfTool, uint256 defaultIdMonster) public
    returns (bool){
3     // typeOfTool 0 => normal tool
4     // typeOfTool 1 => mega tool
5     // typeOfTool 2 => ultra tool
6
7     bool checkUserHaveTool = false;
8     if (typeOfTool == 0 && getNumberOfNormalCaptureToolOfUser() > 0) {
9         checkUserHaveTool = true;
10    } else if (typeOfTool == 1 && getNumberOfMegaCaptureToolOfUser() > 0) {
11        checkUserHaveTool = true;
12    } else if (typeOfTool == 2 && getNumberOfUltralCaptureToolOfUser() > 0)
13    {
14        checkUserHaveTool = true;
15    } else {
16        return false;
17    }
18
19    if (checkUserHaveTool = true) {
20        uint256 randomResultToCapture = (randomResult % 10).add(1);
21        randomResult = 0;
22
23        if (typeOfTool == 0) {

```

```

23         addressToCaputeTools[msg.sender].normal = addressToCaputeTools[
msg.sender].normal.sub(1);
24         if (randomResultToCapture < 4) {
25             addFindableMonsterToUser(defaultIdMonster);
26             emit CaptureResult(true);
27         } else {
28             emit CaptureResult(false);
29         }
30     } else if (typeOfTool == 1) {
31         addressToCaputeTools[msg.sender].mega = addressToCaputeTools[msg
.sender].mega.sub(1);
32         if (randomResultToCapture < 7) {
33             addFindableMonsterToUser(defaultIdMonster);
34             emit CaptureResult(true);
35         } else {
36             emit CaptureResult(false);
37         }
38     } else if (typeOfTool == 2) {
39         addressToCaputeTools[msg.sender].ultra = addressToCaputeTools[
msg.sender].ultra.sub(1);
40         if (randomResultToCapture < 10) {
41             addFindableMonsterToUser(defaultIdMonster);
42             emit CaptureResult(true);
43         } else {
44             emit CaptureResult(true);
45         }
46     } else {
47         emit CaptureResult(false);
48     }
49 } else {
50     emit CaptureResult(false);
51 }
52 }

```

The above function gets called everytime a user uses a catching tool. In particular a random number is utilized that goes from 1 to 10. Based on the obtained number and to the catching tool chosen, it is defined whether the monster will be captured or not.

-	Success Capture	Failed Capture
Normal	1,2,3	4,5,6,7,8,9,10
Mega	1,2,3,4,5,6	7,8,9,10
Ultra	1,2,3,4,5,6,7,8,9	10

3.2.5 Combatti

```

1 // Function to get a random user to fight

```

```

2 function getRandomUser() public view returns(address){
3     address randomUserAddress;
4     if (users.length > 1) {
5         randomUserAddress = users[randomResult % users.length];
6         if (randomUserAddress == msg.sender){
7             if ((randomResult % users.length) == users.length-1){
8                 uint random = (randomResult % users.length) - 1;
9                 randomUserAddress = users[random];
10            } else {
11                uint random = (randomResult % users.length) + 1;
12                randomUserAddress = users[random];
13            }
14        }
15        return randomUserAddress;
16    } else {
17        // Impossible to fight there are not other players on the platform
18        return msg.sender;
19    }
20 }

```

The moment a user wants to start a fight, the above function gets called that allow to find a random monster between the users.

```

1 // Function to get random move for opponent monster
2 function getRandomMove(address randomUserAddress) private view returns(Move
    memory){
3     Move[] memory randomUserActiveMonsterMoves= idToMovesOfPersonalMonsters[
    addressToUser[randomUserAddress].activeMonster];
4     return randomUserActiveMonsterMoves[randomResult %
    idToMovesOfPersonalMonsters[addressToUser[randomUserAddress].activeMonster
    ].length];
5 }

```

The above code allows to select a random attack between the "known" attacks of the opponents active monster.

```

1 // function to define Winner
2 function defineWinner(address randomUserAddress, uint moveSelectedByUser)
    public returns(bool){
3     // return true => User win
4     // return false => User loss
5
6     Move memory randomUserMove = getRandomMove(randomUserAddress);
7     Move[] memory allUserMove = idToMovesOfPersonalMonsters[addressToUser[
    msg.sender].activeMonster];
8     Move memory userMove = allUserMove[moveSelectedByUser];
9
10    uint256 userMonsterHealtResidue = subHealthPoints(msg.sender,
    randomUserMove);
11    uint256 RandomUserMonsterHealtResidue = subHealthPoints(randomUserAddress
    ,userMove);

```

```

12
13     if (userMonsterHealtResidue > RandomUserMonsterHealtResidue) {
14         updateWinningScore(msg.sender);
15         updateLosingScore(randomUserAdress);
16         emit FigthResult(true);
17     } else {
18         updateWinningScore(randomUserAdress);
19         updateLosingScore(msg.sender);
20         emit FigthResult(false);
21     }
22 }

```

The above allows to define the winner of the battle.

3.2.6 Shop

```

1 // Function to buy ultra capture tool
2 function buyUltraCaptureTool() internal {
3     addressToCaputeTools[msg.sender].ultra = addressToCaputeTools[msg.sender
4     ].ultra.add(1);
5 }
6 // Call function to buy capture tools
7 function buyCaptureTool(uint typeOfTool) public payable{
8     if (typeOfTool == 0){
9         buyNormalCaptureTool();
10    } else if (typeOfTool == 1){
11        buyMegaCaptureTool();
12    } else if (typeOfTool == 2){
13        buyUltraCaptureTool();
14    }
15 }

```

The above code, allows the user to buy catching tools. In particular, the first function allows to buy an *Ultra catching tool*.

```

1 // Function that increase the monster's level by 1 and health by 20.
2 function levelUp(uint256 monsterGlobalId) public payable returns (bool) {
3     bool isOwnerOdMonster = checkIsOwner(monsterGlobalId);
4     if (isOwnerOdMonster == true) {
5         addressToMonster[msg.sender][monsterGlobalId].level =
6         addressToMonster[msg.sender][monsterGlobalId].level.add(1);
7         addressToMonster[msg.sender][monsterGlobalId].healthPoints =
8         addressToMonster[msg.sender][monsterGlobalId].healthPoints.add(20);
9         return true;
10    } else {
11        return false;
12    }
13 }

```


The above code allows to increase the level and the life points of the owned monster.

```
1 // Function to sell a monster
2 function sellUserMonster(uint256 idMonster, uint256 price) public
   checkIsNotActiveMonster(idMonster) returns(uint) {
3     monstersOnSell.push(SellMonster(idMonster, price, msg.sender));
4     return monstersOnSell.length;
5 }
```

The above code allows the user to sell an owned monster.

```
1 // Function to buy a monster on sell
2 function buyMonster(uint256 idMonster) public returns (bool) {
3     for (uint256 i = 0; i < monstersOnSell.length; i++) {
4         if (monstersOnSell[i].idMonster == idMonster) {
5             addToBlockchain(
6                 payable(monstersOnSell[i].owner),
7                 monstersOnSell[i].price
8             );
9             for (uint256 j = 0; j < allMonsters.length; j++) {
10                if (allMonsters[j].id == idMonster) {
11                    addressToMonster[msg.sender].push(allMonsters[j]);
12                    for (uint256 k = 0; k < addressToMonster[monstersOnSell[
13i].owner].length; k++) {
14                        if ( addressToMonster[monstersOnSell[i].owner][k].id
15== idMonster ) {
16                            delete addressToMonster[monstersOnSell[i].owner
17][k];
18                        }
19                    }
20                }
21                delete monstersOnSell[i];
22                return true;
23            }
24        }
25        return false;
26 }
```

The above functions allows the user to buy a monster sold by another user.

3.2.7 Admin-executable operations

```
1 //Function where the owner can withdraw from the contract because payable
   was added to the state variable above
2 function withdraw(uint256 _amount) public onlyOwner {
3     Owner.transfer(_amount);
4 }
```

```

5
6 // Function to get the balance of contract
7 function getAmount() public view onlyOwner returns (uint256) {
8     return address(this).balance;
9 }

```

The above functions are accessible only to the admin and allow to visualize the Matic quantity available on the contract and to withdraw them (transferring them to his personal wallet).

3.3 Web3 Javascript

Web3.js is a collection of libraries that allow to interact with a local ethereum node or even a remote node using HTTP, IPC or Websocket. We used this library as it allows us to call functions created in Solidity on the client-side. We will now explain how the connection was made to the wallet.

```

1 const connectWallet = async () => {
2   let provider = window.ethereum;
3   const web3 = new Web3(provider);
4
5   if (typeof provider !== "undefined") {
6     provider
7       .request({ method: "eth_requestAccounts" })
8       .then((accounts) => {
9         console.log("Herer", accounts);
10        setCurrentAccount(accounts[0]);
11
12        web3.eth
13          .getBalance(accounts[0])
14          .then((res) =>
15            setMaticBalance(web3.utils.fromWei(res, "ether").substring(0, 5))
16          );
17      })
18      .catch((err) => {
19        console.log(err);
20      });
21
22   window.ethereum.on("accountsChanged", function (accounts) {
23     setCurrentAccount(accounts[0]);
24   });
25 }

```

4. Screens

4.1 Homepage


LOGO Owner Profile Play Shop Change wallet 0x963...2f0b			
Account	Level	Victories/Losses	Active monster
Karan 0x963...2f0b	1	0/0	Grassmonster  Lvl. 1

Figure 4.1: Homepage

4.2 Profile

LOGO

Owner


Profile

Play





Shop

Change wallet0x963...2f0b

2

Account	Balance	Level	Victories/Losses
Karan 0x963...2f0b	0.110 	1	0/0

Your Monsters

Name	Type	Health Points	Level	Image
Grassmonster (Active)	Leaf 	90	1	
Firedragon	Fire 	100	1	

Your snipping tools




Name	Image	Probability	Owned
Normal		30%	2
Mega		60%	0
Ultra		90%	2

Figure 4.2: User Profile

4.3 Play

4.3.1 Catch

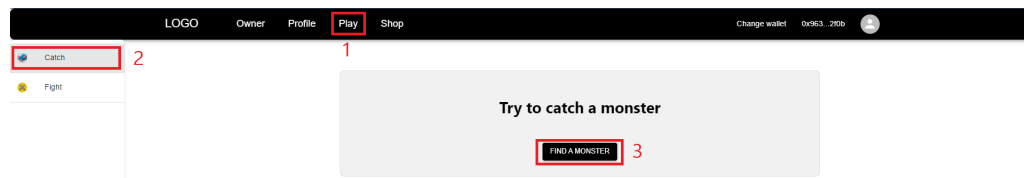


Figure 4.3: Catch monster

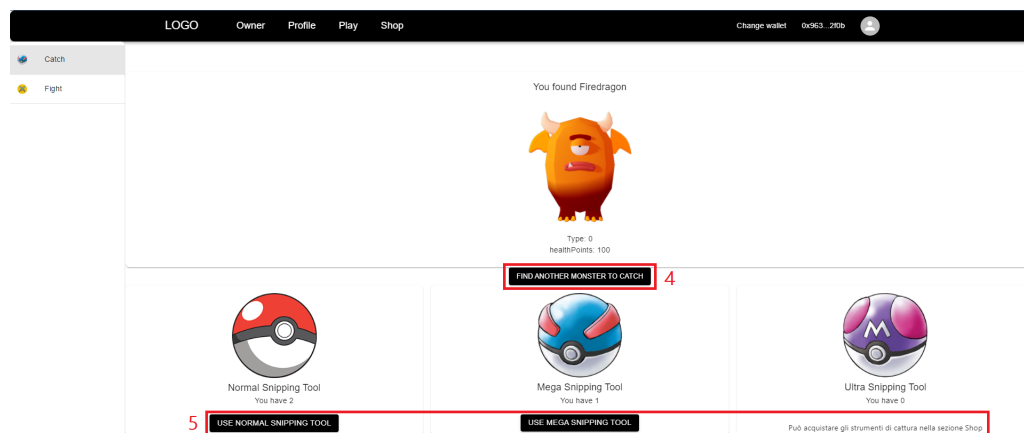


Figure 4.4: Catch monster

4.3.2 Fight



Figure 4.5: Combattimento contro altri utenti

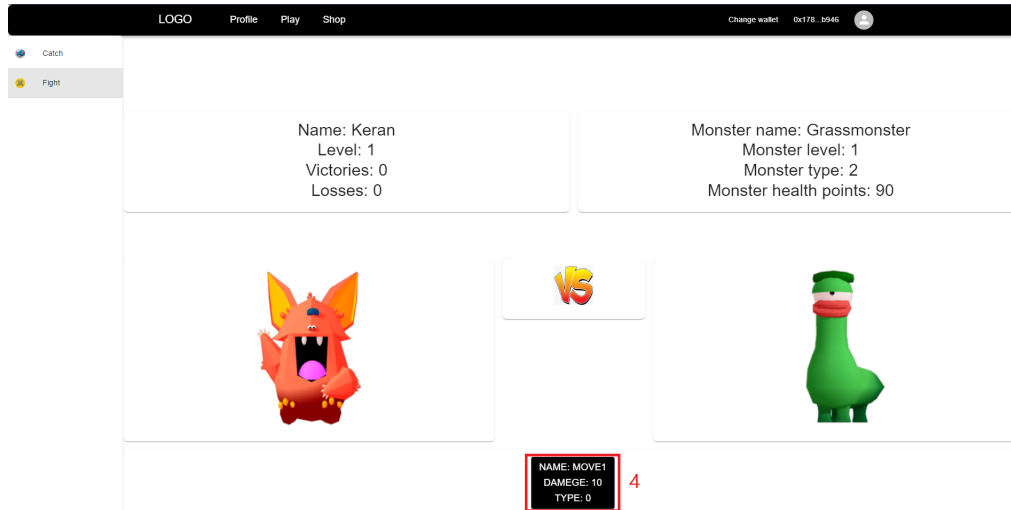


Figure 4.6: Fight against other players

4.4 Shop

4.4.1 Snipping Tools

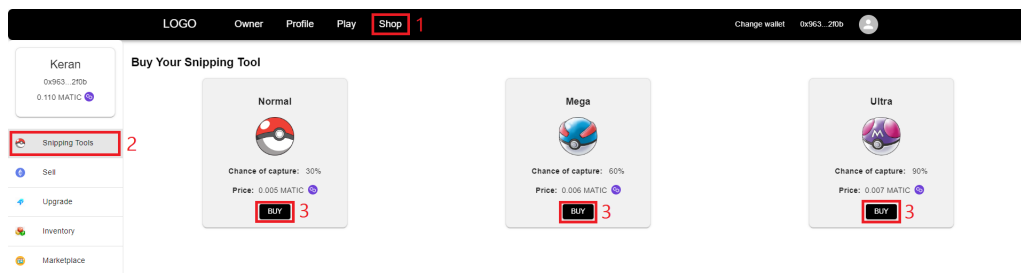


Figure 4.7: Buy catching tools

4.4.2 Sell

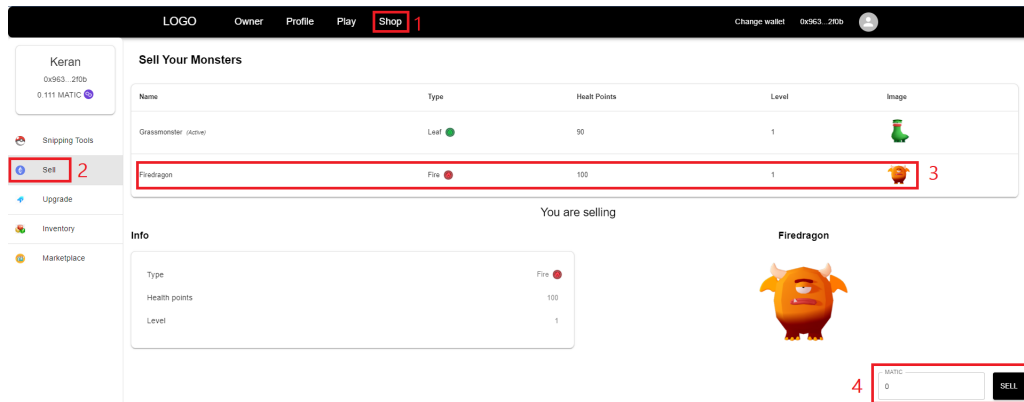


Figure 4.8: Sell a monster

4.4.3 Upgrade

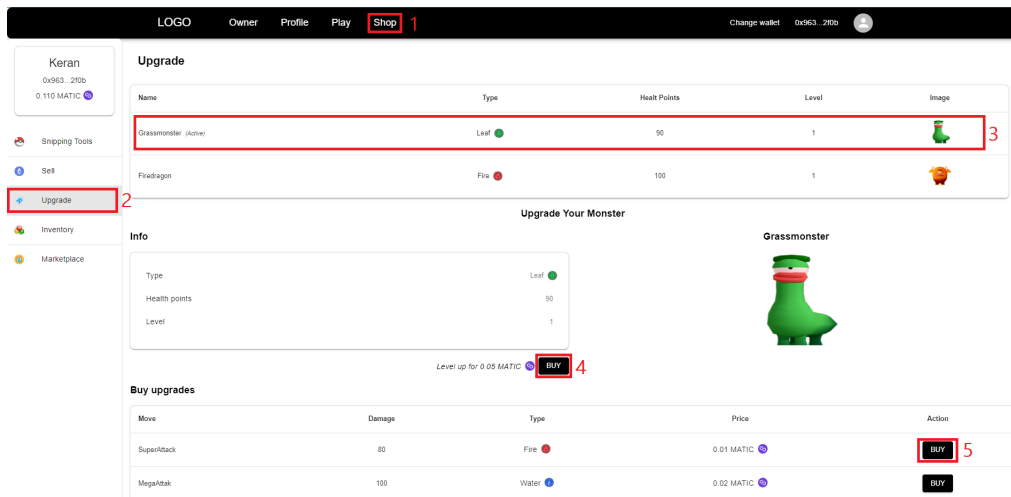


Figure 4.9: Upgrade your monster

4.4.4 Marketplace

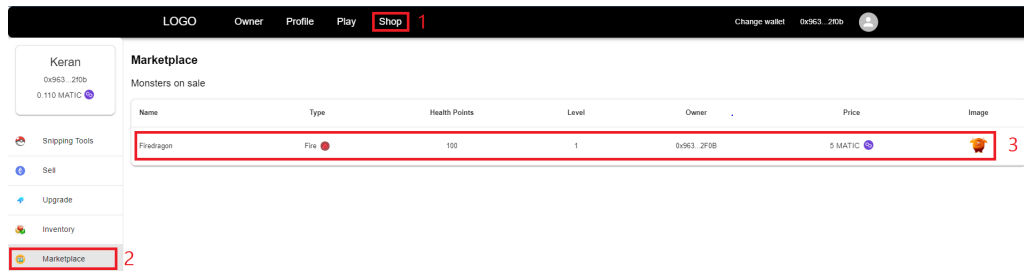


Figure 4.10: Visualize all the monsters on sale

4.5 Owner

4.5.1 Add new monster

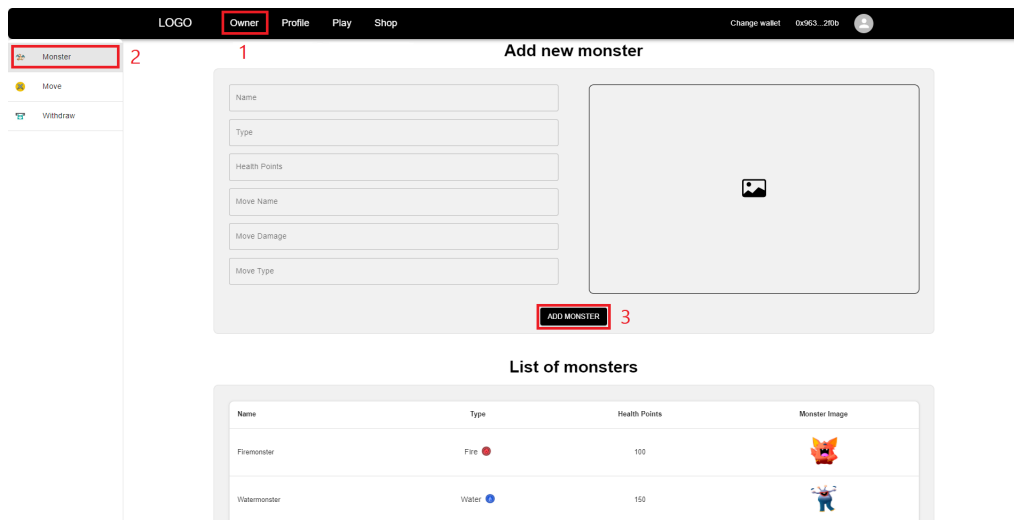
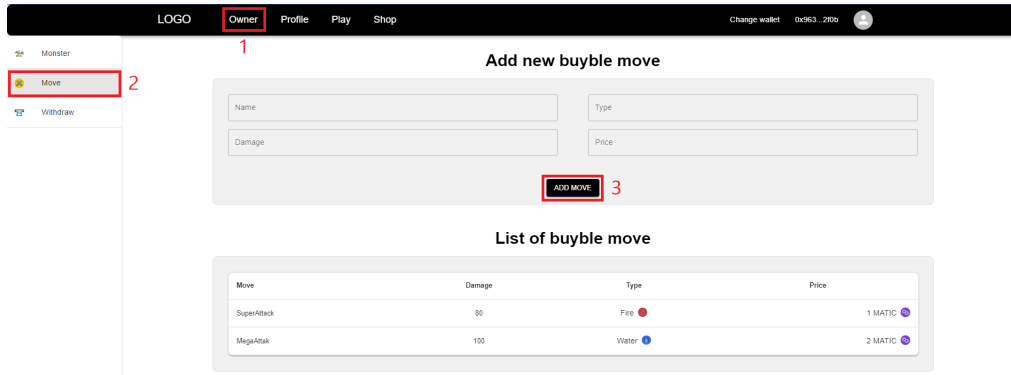


Figure 4.11: Admin - add new monster

4.5.2 Add new move



1

2

3

Add new buyable move

Name:

Type:

Damage:

Price:

ADD MOVE

List of buyable move

Move	Damage	Type	Price
SuperAttack	80	Fire	1 MATIC
MegaAttack	100	Water	2 MATIC

Figure 4.12: Admin - add new attack

4.5.3 Withdraw



1

2

3

4

5

Withdraw

Amount:

GET AMOUNT

MATIC:

WITHDRAW

Figure 4.13: Admin - withdraw Matic from the contract