MLH localhost

Workshop

The Basics of Blockchain with Horizen

MLH localhost

HORIZEN™

v.0.0.1

Our Mission is to Empower Hackers.

65,000+
HACKERS

12,000+
PROJECTS CREATED

400+
CITIES

We hope you learn something awesome today!
Find more resources: http://mlh.io/

Table of Contents

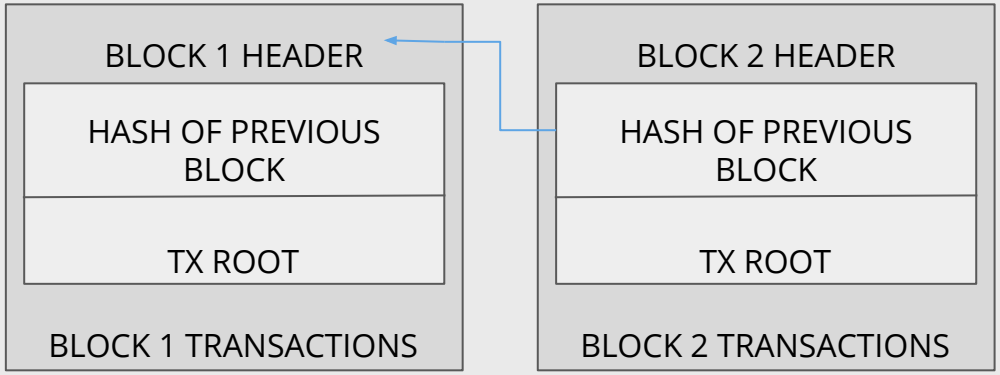
What will you learn today?

1. An Introduction to Blockchain technology and Horizen
2. How to create a Sphere wallet to send and receive transactions
3. How to deploy and run an application that uses the Horizen Blockchain

When you hear the word ‘Blockchain’, what do you think about?

What is Blockchain?

A blockchain is a **data structure**. Common examples of data structures are lists or tables.



Key Term

Data structure: Stores data in a structured format.

The blockchain is a **chain of "blocks"**

Each block is a collection of data, similar to a folder on your desktop.

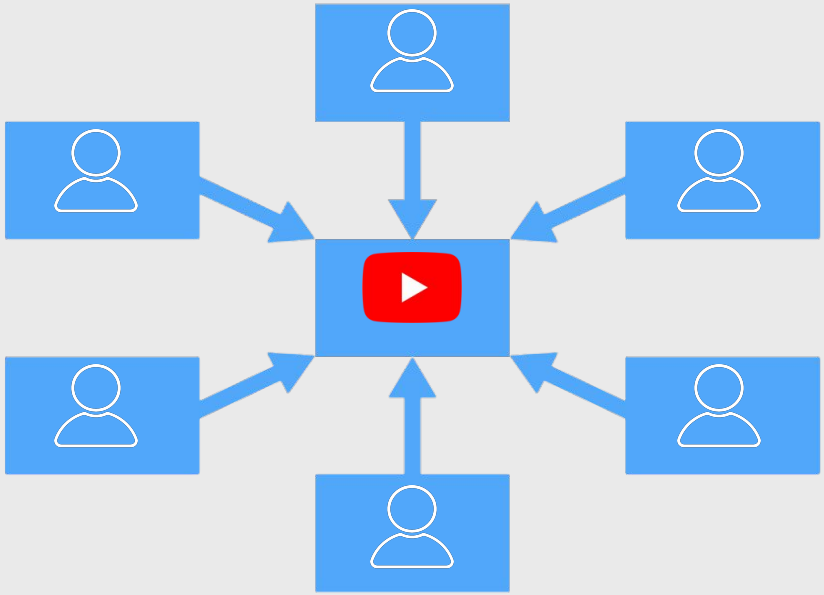
Each block knows what block came before it, thus forming a chain.

Centralized Systems

Most data is stored in databases, e.g. your user data on Facebook, your order history on Amazon or your videos on YouTube.

Databases are highly efficient, but **centralized**.

A central entity decides who can add data to DB, who can modify it, and who can delete data from it.



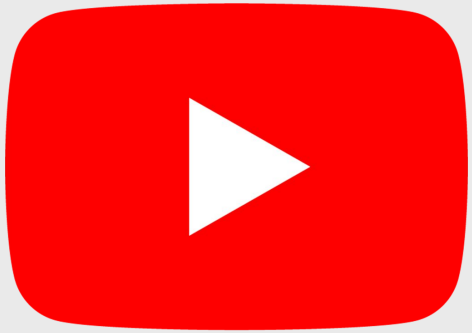
Can you see any problems with storing data in databases?

Centralized Systems

Let's think about YouTube, which has a database to store all your videos.

- What happens when YouTube decides to delete your video because it thinks it is inappropriate?
- What will happen if YouTube shuts down?

In both cases, we lost access to the videos that we created or loved!



Blockchain as a Solution

Blockchain was invented in 2008 to help solve the problems that can arise from having a single, centralised source of knowledge.

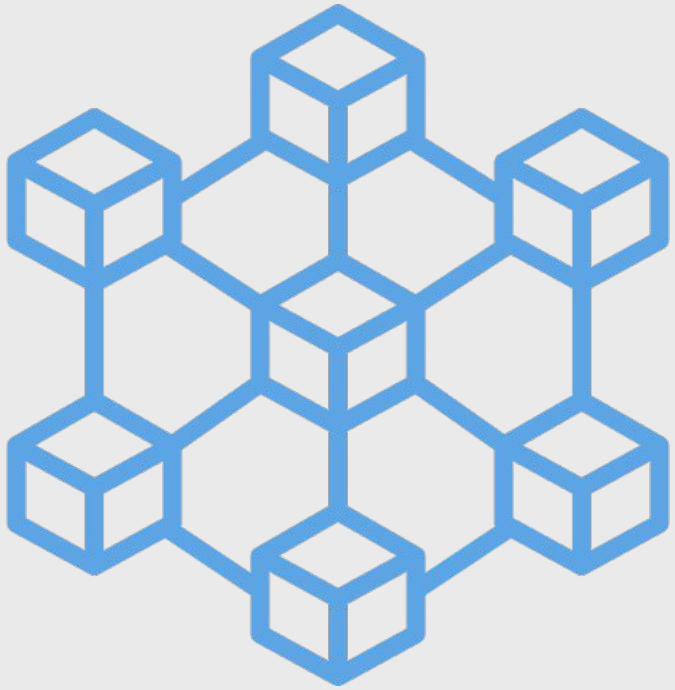
It sought to create a **decentralized** platform: one where there was no single source of truth, and all knowledge was shared **collectively**.

Blockchain as a Solution

Again, let's go back to our YouTube example.

What if....

- Everyone had a copy of the all the videos created.
- All new videos are verified by everyone's YouTube video copy.
- Only the rightful owners of a video are able to edit or delete it.
- Videos are sent directly to one another (**peer to peer**), instead of through a central application like YouTube.



This is the foundation of Blockchain!

A History of Blockchain

The key to Blockchain is **cryptography**. Each new block in the ledger is linked to the previous one with a highly secure **cryptographic hash**, making them incredibly difficult to modify.

Blockchain is *a chain of securely connected blocks of data*.

Key Term

Cryptographic Hashfunction: An algorithm creating a one-way unique fingerprint (hash) of a file.

Blockchain technology took off in the 2008 when an unknown person(s) under the psuedonym Satoshi Nakamoto used Blockchain technology to create the cryptocurrency **Bitcoin.**

What can we use Blockchain for?

Digital currency is one major use of the Blockchain, but definitely not the only one!

Governance: Bring full transparency to elections by making the results fully transparent and publicly accessible.

Healthcare: Private health records can be encrypted and securely transferred.

Music: Blockchain can provide automated handling of royalty payments.

What can we use Blockchain for?

One of the biggest uses of Blockchain is to enable the transfer of digital currency, this is called **cryptocurrency**.

Cryptocurrencies use **cryptography** to provide secure online transactions.

The first cryptocurrency was **Bitcoin**, however there are over 2000 cryptocurrencies currently in the market.

ZEN is the native cryptocurrency of Horizen.

Why Cryptocurrency?

Are there any benefits to using cryptocurrencies rather than normal currencies like the dollar?

Since they leverage blockchains, the transfer of cryptocurrencies is

- Decentralized
- Secure
- Available 24*7
- Global: Transferring cryptocurrency is much cheaper for international transfers than regular currencies.
- Can be anonymous: ZEN provides special addresses called *shielded addresses*, transactions between which are completely private and anonymous.

VS

Introducing Horizen

Horizen is an inclusive ecosystem with the following components -

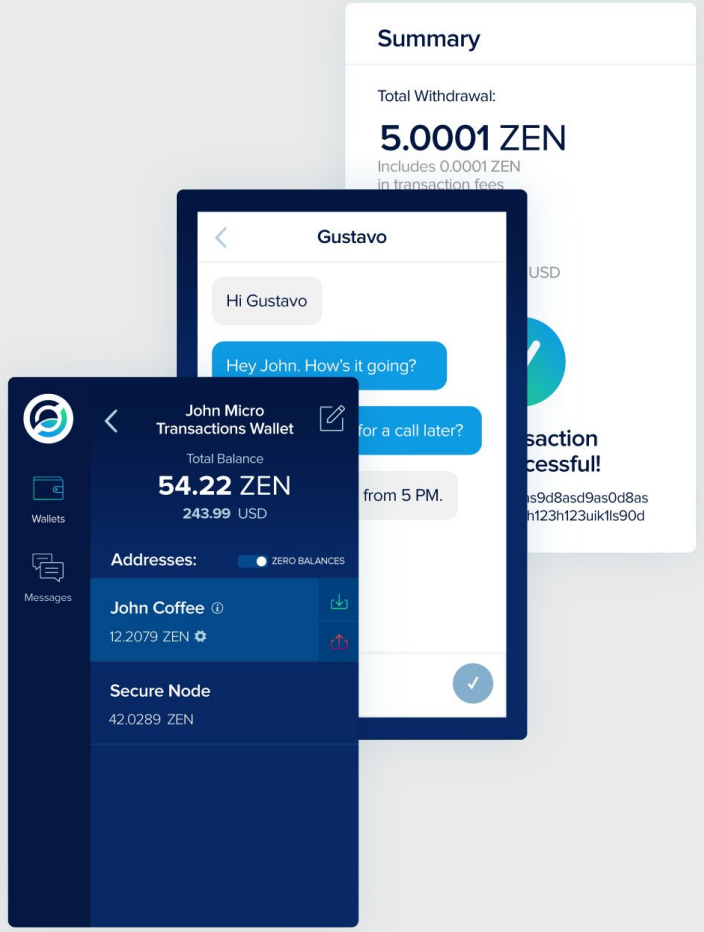
- **ZEN**: The cryptocurrency with full end-to-end encryption to protect internode communication.
- **Infrastructure**: A robust network of 38916 nodes that form the Horizen main blockchain (mainchain).
- **Sidechain technology**: A solution that allows scalability of blockchains by allowing application specific blockchains that can interoperate with one another.
- **Apps, Products and Services** built on Horizen's infrastructure and technology, such as -
    - Zenchat - a messaging app that provides high level of privacy.
    - Sphere - a multifunctional app that also acts as a wallet for ZEN.

Introducing Sphere

Sphere is a multifunctional app created by Horizen.

It's a desktop based tool that's going to allow us to create a ZEN wallet and send and receive money from it.

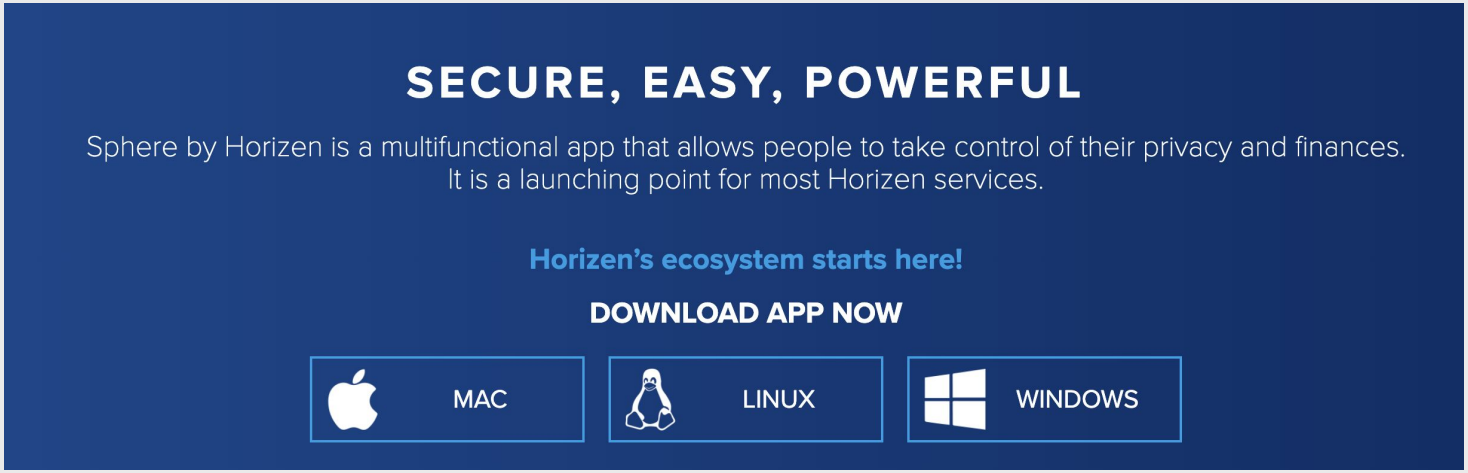
Let's get started!



Download Sphere

Choose the download most appropriate for your OS.

mlhlocal.host/sphere



Install Sphere

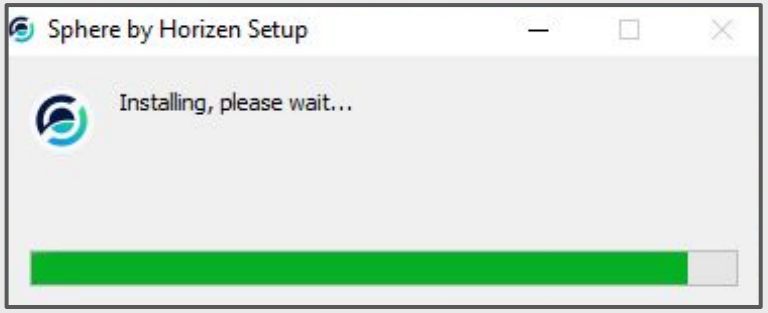
MAC

Once the dmg file is downloaded, double click on it.

Drag and drop the icon into the Applications folder to complete the installation.

WINDOWS

Once the exe file is downloaded, double click on it to get started with the installation.



Elements of a Blockchain

There are 2 main elements of a blockchain -

- **Nodes**
  - Network of computers that form the blockchain.

- **Miners**
  - Nodes that are creating new blocks.
  - Miners have to solve a cryptographic puzzle to create a valid block.
  - Miners get some newly issued cryptocurrency for creating a block.
  - Every cryptocurrency out there started as the block reward for some miner.

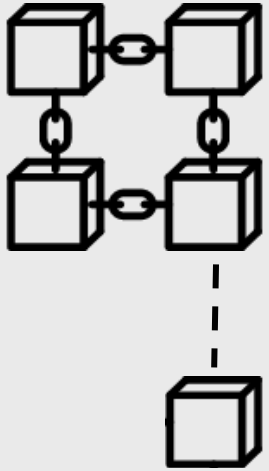
How is Information Shared on a Blockchain?

Whenever a node receives new information, it will share that information with a few known peers.

That node then shares the information with a few of its known peers. This continues until all the information is passed onto all the connected nodes.

This is known as the **Gossip protocol**.

Every message is signed by its sender, thereby allowing nodes sending fake information to be easily identified and the distribution of the information(s) to unwanted targets to be prevented.



Identity on the Blockchain

Blockchain operates on the concept of asymmetric or public key cryptography.

In **asymmetric key cryptography**, the data is encrypted with one key and decrypted with the other key. The keys always come in pairs.

Public key is a user's public identity. People can use this to encrypt information they wish to share privately with them.

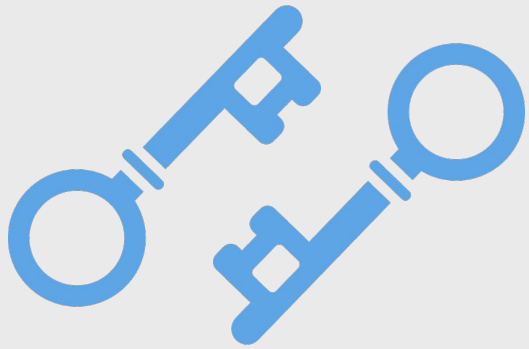
Identity on the Blockchain

Only the user can see the original information once it is encrypted, because only they have access to their private key.

The user's **identity** on the blockchain is their public and private key.

For the user to prove that they sent a message, they can digitally *sign* it using their private key.

Anyone who has access to the user's public key can verify this *signature* and confirm that they did in fact send it.



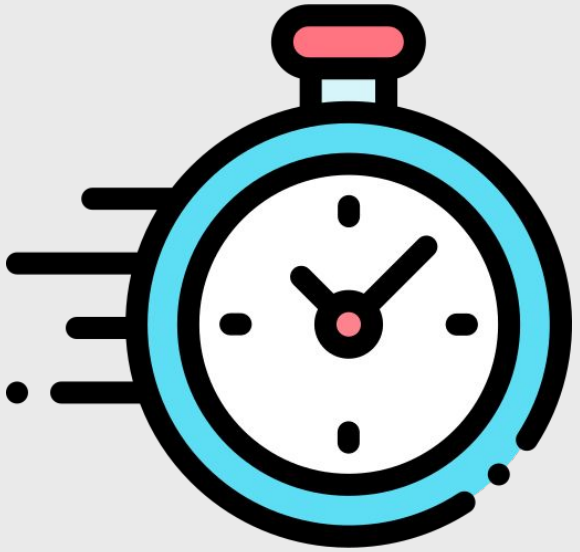
Key Term

Digital Signature: A mathematical scheme for verifying the authenticity of digital messages.

Table of Contents

Scalability Problem of Blockchains

Today most blockchains can only process a handful of transactions per second, while payment networks like VISA support several thousand transactions per second.



Blockchains need to become more scalable to support mainstream adoption.

What is the solution?

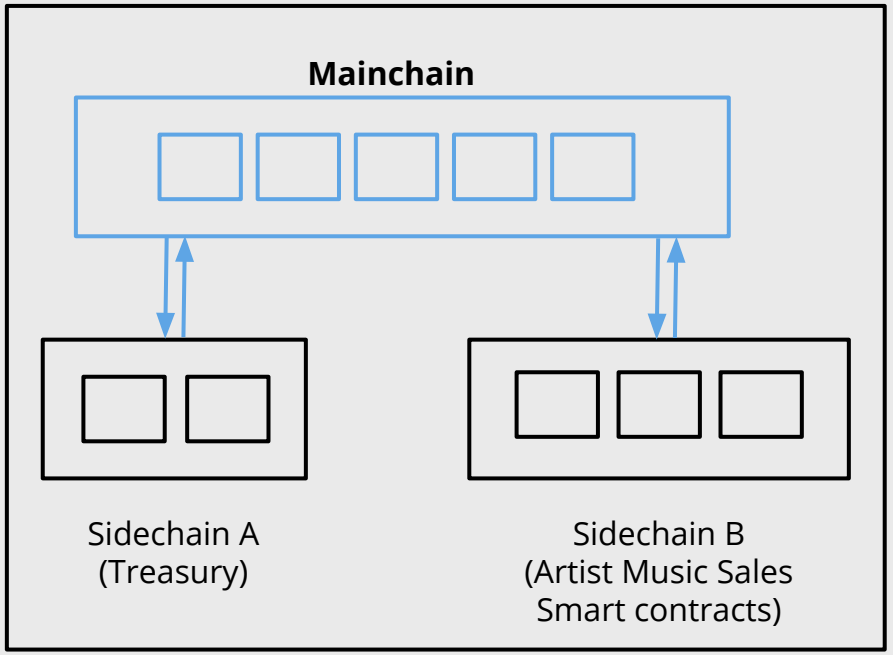
Sidechains as a Solution

A **sidechain** is a full blockchain in itself, with the added feature of being interoperable with the main blockchain.

Users can tweak the features of a single chain to satisfy their needs, without affecting the other chains.

For example. Sidechain A can be for a treasury, and Sidechain B can be for supporting the smart contracts for artist music sales.

These can interact with each other and the main blockchain when required.

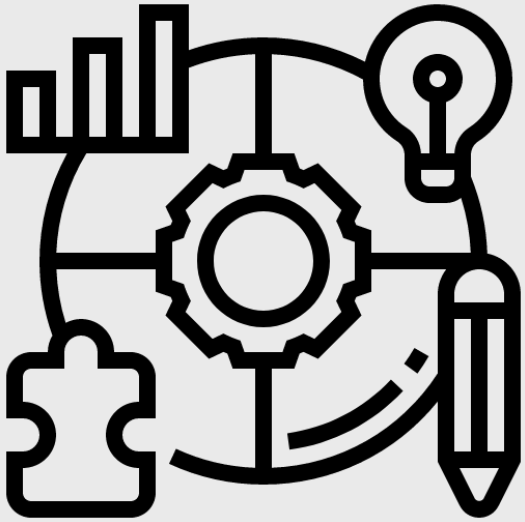


How can you use sidechains?

Since each sidechain is a full blockchain, it requires the implementation of blockchain code.

Horizen provides a Sidechain SDK with all the necessary components required for building a blockchain in a single toolbox.

All you need to care about is the application specific code - so now you can get started on your application faster than ever!



What will you learn today?

What will you learn today?

We're going to deploy a Pong application where users bet ZEN to play the game, and will either gain or lose ZEN based on the outcome.

To create the application we will work through:

- Launching the Pong application
- Setting up a ZEN wallet
- Interacting with the Horizen blockchain.

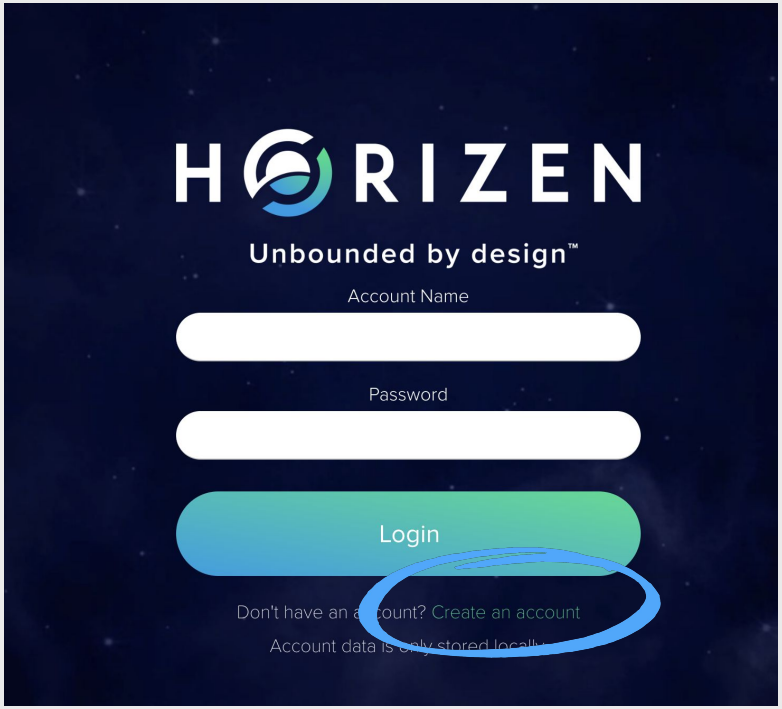
Table of Contents

Creating a wallet

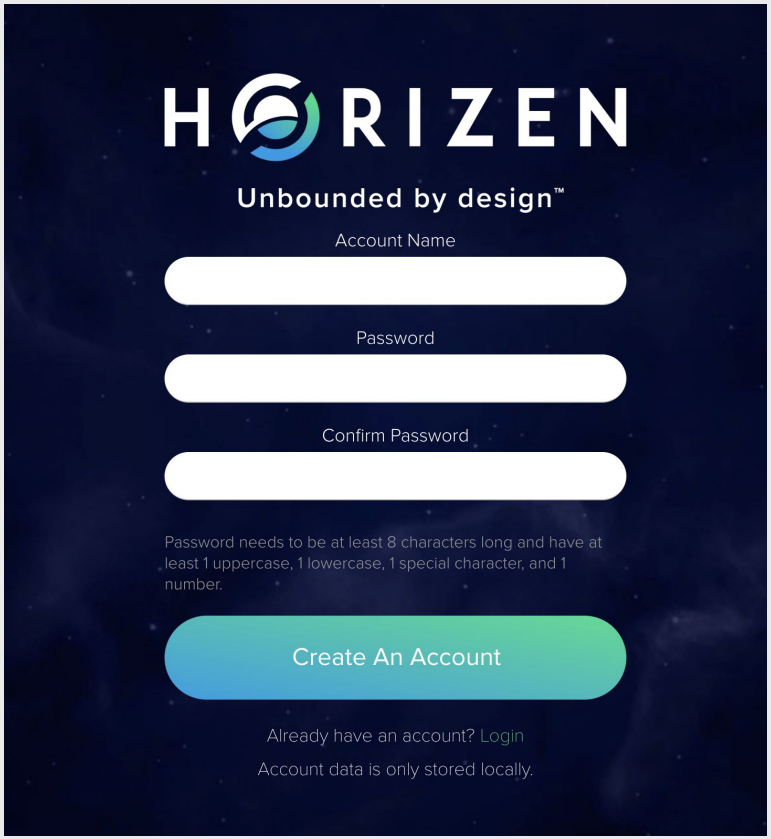
Once the application is installed, open it up.

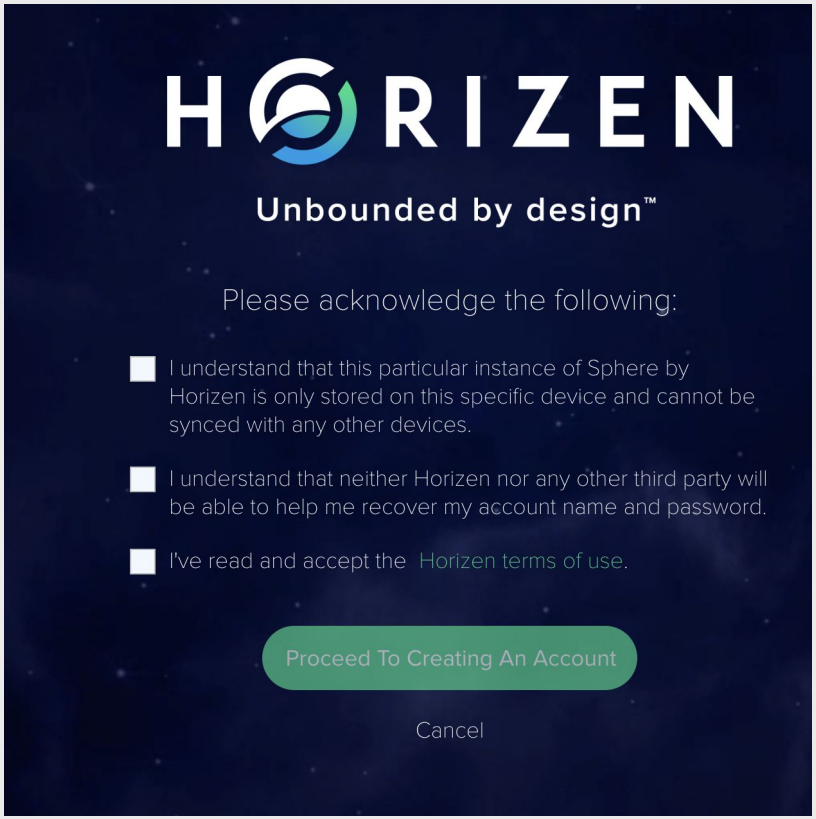
You will see the following page.

Click on create a new account.



Creating a wallet



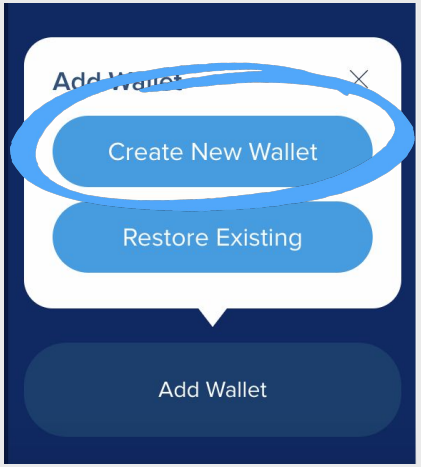


Choose an account name and strong password, then click “Proceed to Creating an Account”

Creating a wallet

You should see the following screen on the left of the application.

Click on “Add Wallet” then click on “Create New Wallet”.

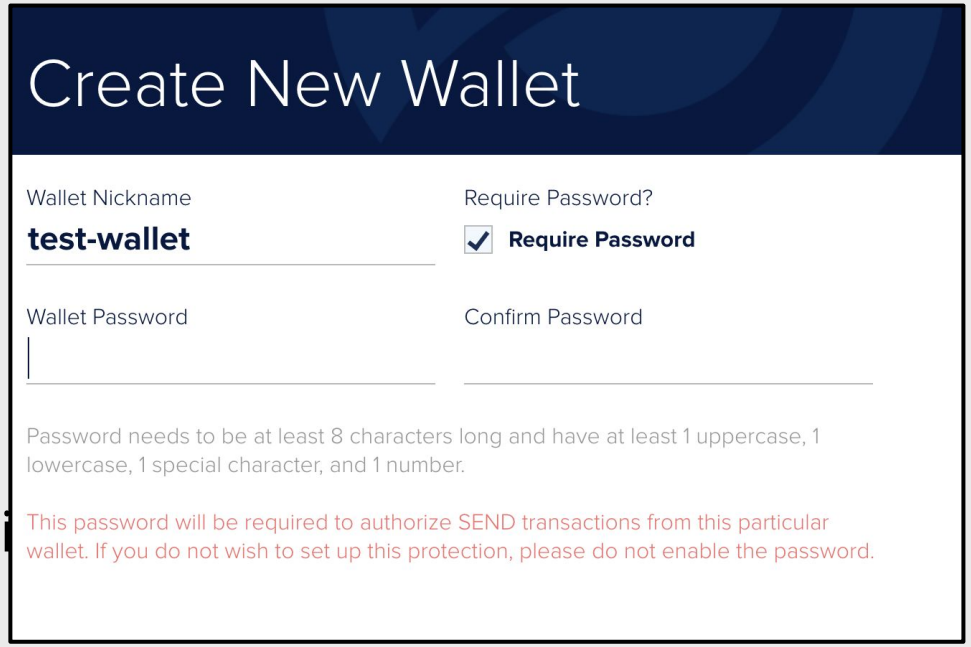


Creating a wallet

Name the wallet “test-wallet”

Click the checkbox that says “Require Password”, and set a strong password.

This password will be required when we try to send ZEN from thi wallet.

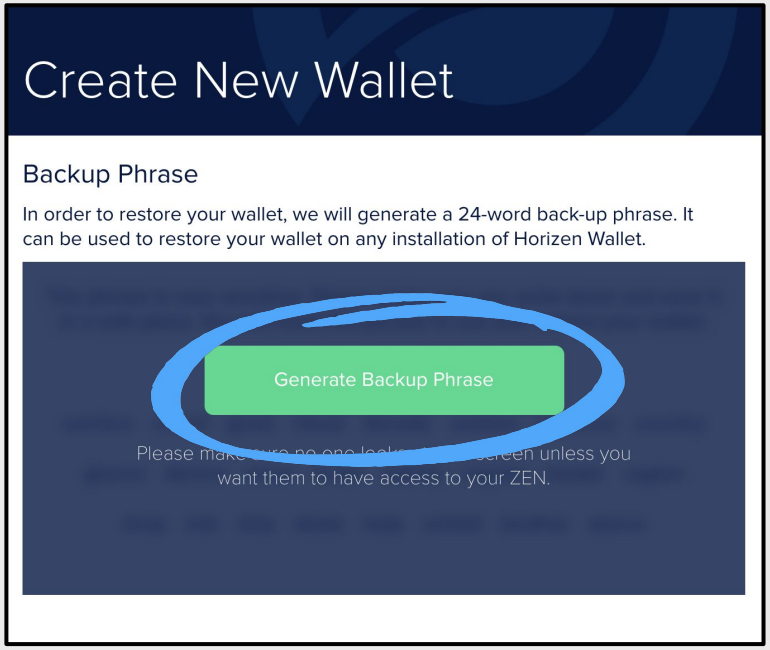


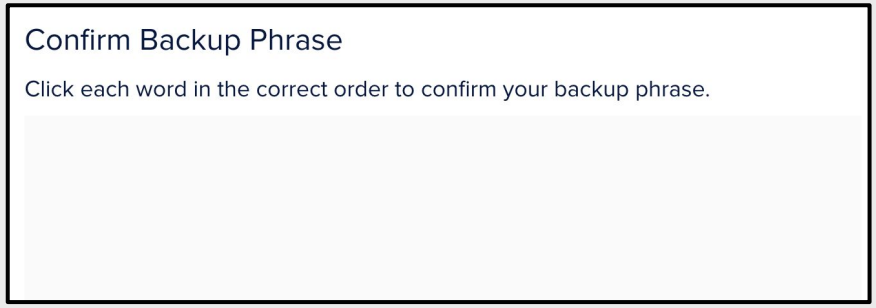
Creating a wallet

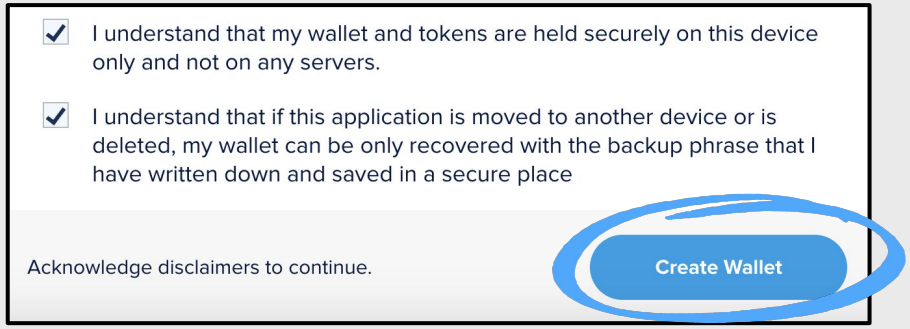
Generate the backup phrase. Copy it somewhere safe.

Enter it in order again.

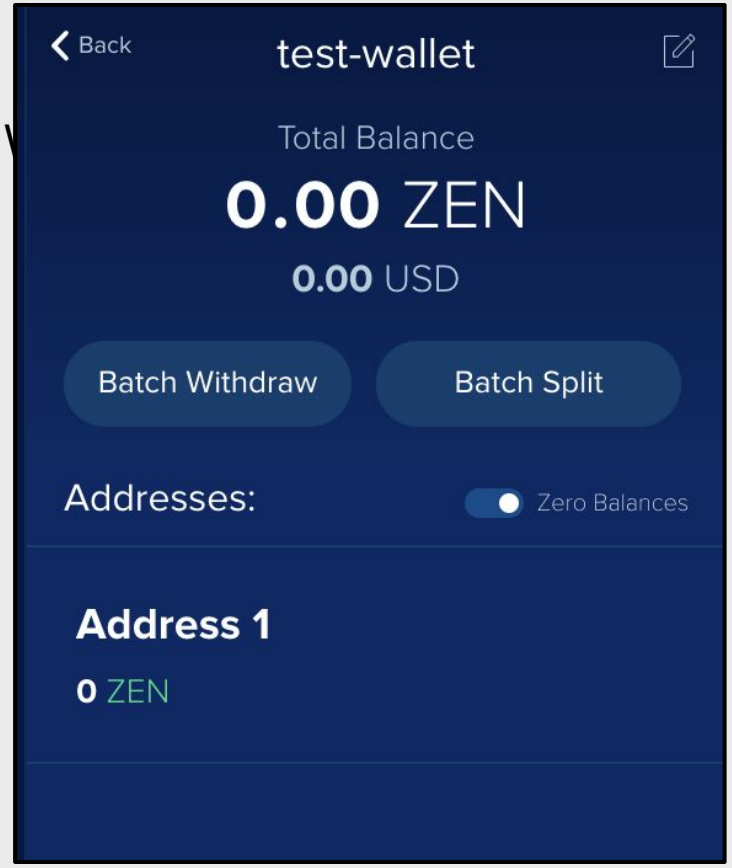
Acknowledge the warnings to complete wallet creation







Creating a wallet



Click on the info button next to Address 1, and copy the address.

Share it with your host to get ZEN in your wallet!

Let’s recap!

We’ve set up Sphere wallet and populated it with some ZEN.

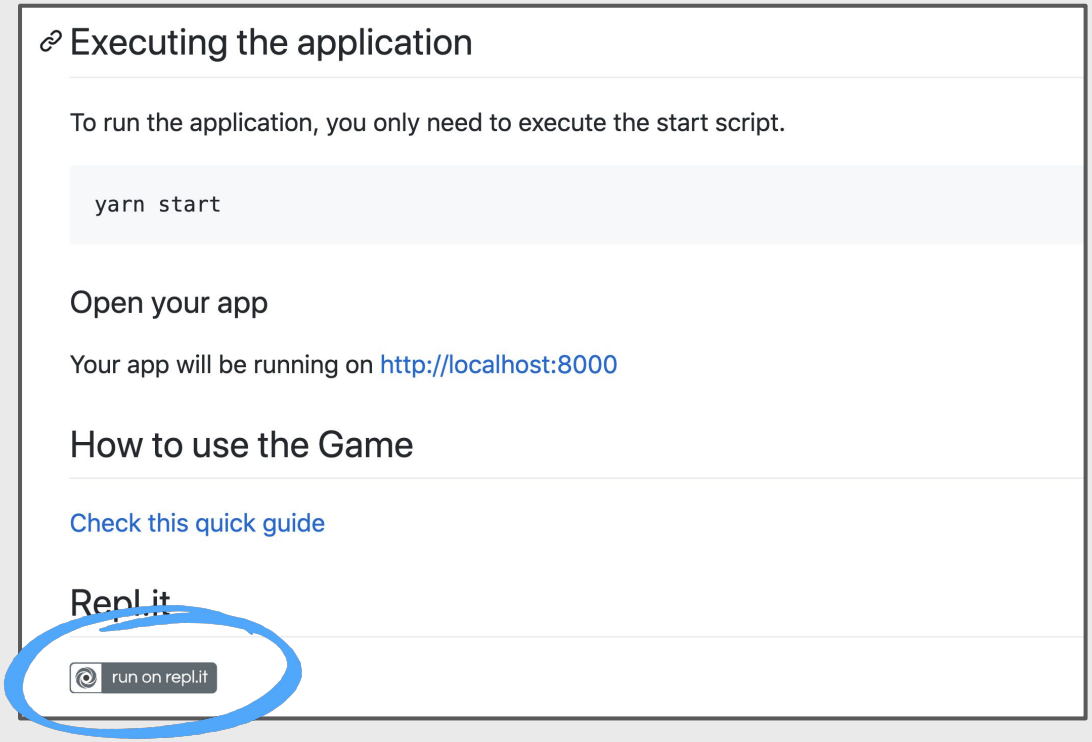
Next, we need to get the source code for the web application.

Get the Source Code

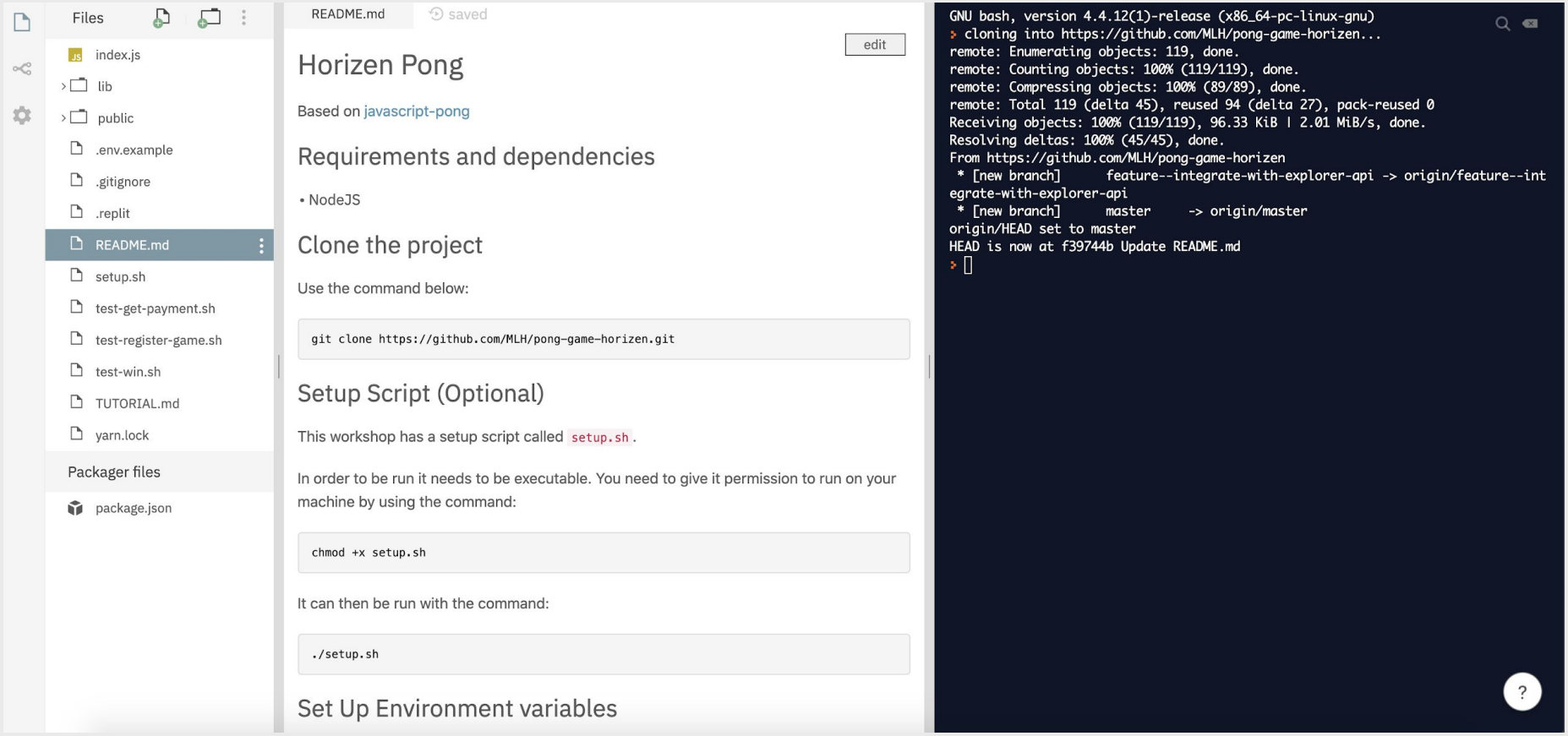
Head to the URL below to see the code on Glitch!

mlhlocal.host/horizen-ping-pong-starter

1. Click on the **run on repl.it** button.
2. View the code on repl.it!



Get the Source Code



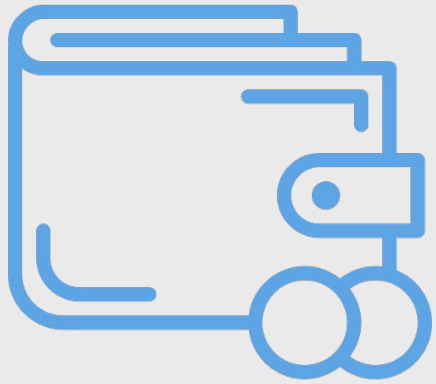
Repl has 3 main windows - the leftmost one **displays the folder structure**, the middle one shows the **code** and the right one is the **terminal**.

Transactions on the Blockchain

In blockchains, **transactions** are messages to the entire network informing it of a transfer of money.

These transactions are written into the blockchain and are therefore, **public** and **unchangeable**.

Cryptographic wallets help in creating transactions.



Key Term

Cryptographic wallet: An application that has 3 main functions - managing your keys, showing you your balance and creating transactions.

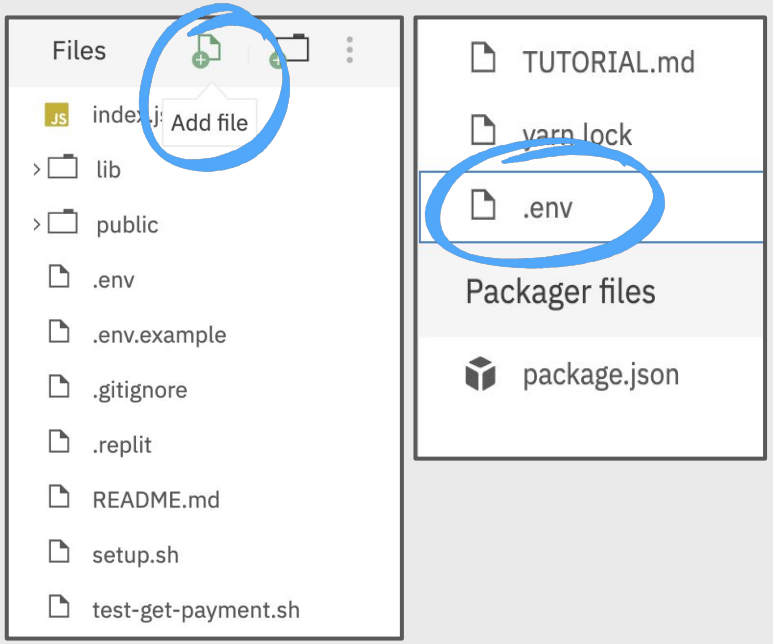
Creating the .env file

Add a new file by clicking on the add file button in repl.

Name it `.env`

Open up the file, type in `ZEN_PONG_WALLET_SEED=`

Then paste in the 24 word backup phrase that you copied from Sphere inside double quotes.



.env

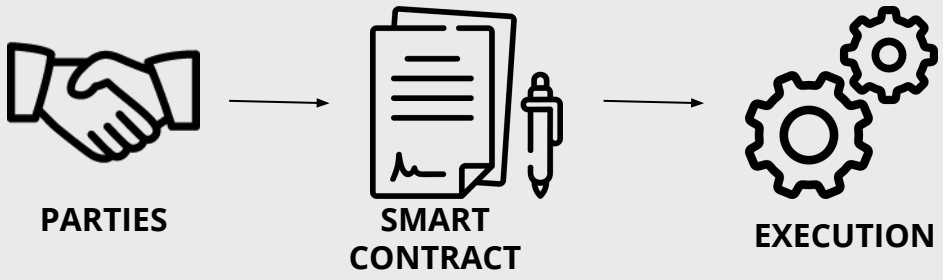
```
ZEN_PONG_WALLET_SEED="seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed seed"
```

Smart Contracts

A smart contract is basically software on a blockchain.

It allows the performance of credible transactions without third parties.

It is a collection of “if..then” statements that will execute based on predefined conditions.



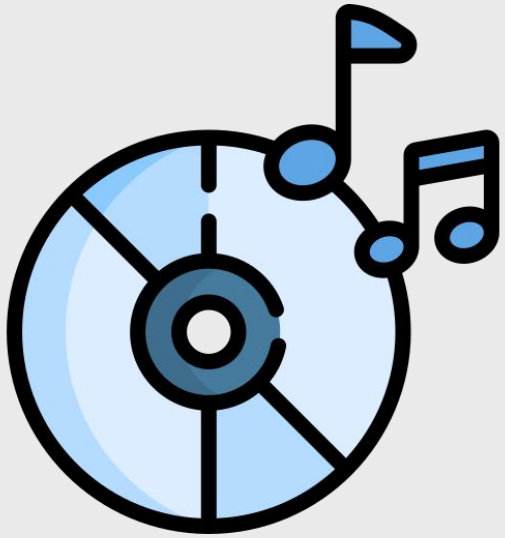
Smart Contracts

Smart contracts can be used anywhere there is a requirement for a middleman.

For example, artists wanting to sell their music usually need to go through a distribution network.

Instead, they could set up a smart contract that releases a copy of their music to anyone who has fulfilled the clause of the smart contract

i.e. transferred the appropriate amount to their cryptocurrency wallet.



Smart Contracts

Smart contracts provide a lot of advantages:

1. **Safety** - smart contracts run on a blockchain, where trust and transparency is built into the system.

2. **Savings** - they avoid the need for a large chain of middlemen.

3. **Accuracy** - less prone to manual errors.

4. **Guaranteed outcomes** - since smart contracts are irreversible and tamper proof, once these self-executing smart contracts are set in motion, the outcome is guaranteed.

Code review

public/scripts/index.js

```
185 startForms() {
186    UI.startGameForm();
187    UI.startPaymentForm();
188    UI.startPrizeForm();
189 }
```

Open up the index.js file which is inside the public/scripts folder.

Lines 186-188 show the 3 main methods that are called when the game is started.

Let us look at what each of these methods do one by one.

Code review

public/scripts/index.js

```
85 startGameForm() {
86         const startGameButton = document.getElementById("start-game-button");
87         startGameButton.addEventListener("click", async event => {
88             event.preventDefault();

90             const registeredGame = await API.createGame();
91             const { gameWallet } = registeredGame;
92             STATE.game = registeredGame;
```

When the user clicks on the Start Game button, the event listener kicks in (**line 87**).

In **line 90**, we create a game by calling the API, then store its details in the `gameWallet` and `STATE.game` objects.

Code review

public/scripts/index.js

```
85 startGameForm() {
          ...
94          UI.updatePaymentInstructions(gameWallet);
95          UI.hideResultsContainer();
96          UI.hideGameContainer();
97          UI.hideGameForm();
98          UI.displayPaymentContainer();
99      });
100 },
```

In **line 94**, we update the payment instructions.

In **line 95-98** we hide every other screen but the payment container.

Code review

public/scripts/index.js

```
129 const watchForPayment = async () => {
130       const transactions = await API.listTransactions(STATE.game.id);
131       const hasPaid = !!transactions.length;

133       if (hasPaid) {
134           checkPaymentButton.textContent = "We will start the game soon.";
135           setTimeout(() => {
136                UI.displayGameContainer();
137                UI.hidePaymentContainer();
138                startGame();
139           }, 1000);
140           return hasPaid;
141       }
```

The `watchForPayment` function is the most important part of the `startPaymentForm` function.

It waits for the notification of a payment to the game wallet. Once it receives the notification, it starts a new game.

Code review

public/scripts/index.js

```
129 const watchForPayment = async () => {
          ...
142       else {
143           checkPaymentButton.textContent = "Waiting for payment...";
144           setTimeout(() => watchForPayment(), 5000);
145       }
146 };
```

If it does not receive a payment, it checks again in 5s (**line 144**).

It will keep waiting for the payment, unless we explicitly quit the program.

Code review

public/scripts/index.js

```
197 async function finishGame() {
198     UI.hideGameContainer();

200     const hasWon = STATE.game.result === 0;
201     if (hasWon) {
202         UI.displayWin();
203     } else {
204         UI.displayLoss();
205     }

207     UI.displayResultsContainer();
208 }
```

Once the game is finished, this function checks whether the user has won or not and displays the corresponding message.

Code review

public/scripts/index.js

```
152 startPrizeForm() {
        ...
173     const playerWallet = playerWalletInput.value;

175     const response = await API.updateGamePlayerWallet(
176       STATE.game.id,
177       playerWallet
178     );

180     const url =
`https://explorer.horizen.global/tx/${response.transaction.txid}`;
181     UI.displaySuccessPrize(url);
182     });
183 },
```

Once the game is complete and the user has won, the `startPrizeForm` function gets triggered, which takes in the user's wallet information, sends ZEN coins to it, and returns the transaction details.

Code review

/index.js

```
27 app.post(
28     "/api/games",
29     wrapAsync(async (req, res) => {
           // Fetch first available address to set as the game's wallet
31         const addresses = await Zen.getWalletAddresses();
32         const firstAddress = addresses[0];
33         const { address: gameWallet } = firstAddress;

           // Creates a new Game in memory
36         const game = Game.registerGame(gameWallet);

38         res.json({ data: { game } });
39     })
40 );
```

Let's look at /**index.js**, which is where most of the logic of the code sits.

To register a game, we first get addresses of the game, then register it using the registerGame method in game.js

Code review

index.js

```
103 let transactions =
104       (await Zen.getTransactions(wallet, { from: playerWallet
})).items.filter(
105         t => {
106           const toAddresses = t.vin.map(v => v.addr);
107           const isRecent = t.time >= timestamp;

109           return all || isRecent;
110         }
111       ) || [];
112     res.json({ data: transactions });
113   })
```

How do we know if the user has actually sent money to the game's wallet? The `/api/games/:id/payments` endpoint takes care of it.

Let us look closely at **lines 103-113**, which is where the magic happens.

Code review

index.js

```
103 let transactions =
104       (await Zen.getTransactions(wallet, { from: playerWallet
}))).items.filter(
105         t => {
106           const toAddresses = t.vin.map(v => v.addr);
107           const isRecent = t.time >= timestamp;

109           return all || isRecent;
110         }
111       ) || [];
112     res.json({ data: transactions });
113   })
```

We get all the transactions of the playerWallet and filter out those transactions that are

1. not to the game's address and
2. that occured before the game started.

If there is any transaction left, we send the details of the transaction to the UI, or we return an empty list.

Code review

index.js

```
59 const game = Game.updatePlayerWallet(id, playerWallet);
   ...
62 if (game.result === Game.RESULT_WIN) {
63     const { gameWallet } = game;

65     // Create transaction for the winner's prize
66     const tx = await Zen.createTransaction(gameWallet, playerWallet);

68     // Add transaction info to the game object
69     game.transaction = tx;

71     res.json({ data: { game } });
72 }
```

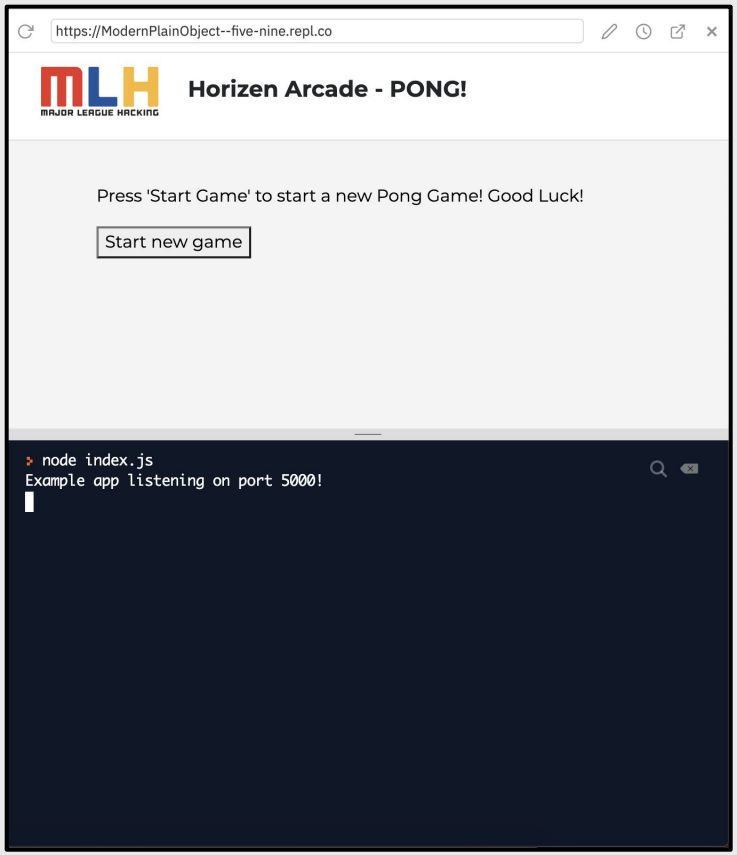
To update the player's wallet after they have won, we first store the player's wallet info in the game object.

Then we create and broadcast a transaction, and store it's details in the game object too.

Launching the WebApp

Let's check out the web application by running it.

Click on the run button in repl.it



Start playing!

Horizen Arcade - PONG!

Press 'Start Game' to start a new Pong Game! Good Luck!

Start new game

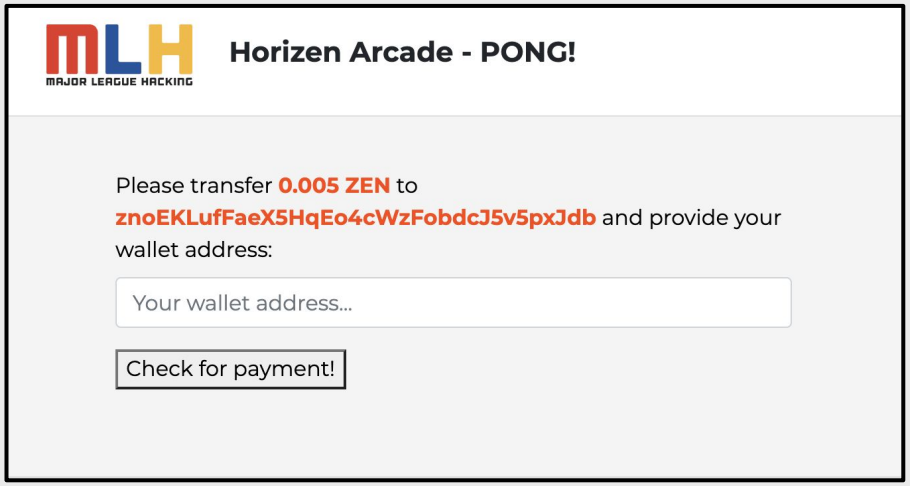
Play the Game

Once you click on Start Game, the application asks you to transfer 0.005 ZEN to a wallet address.

Let's do that now.

Go to Sphere, to the address you created.

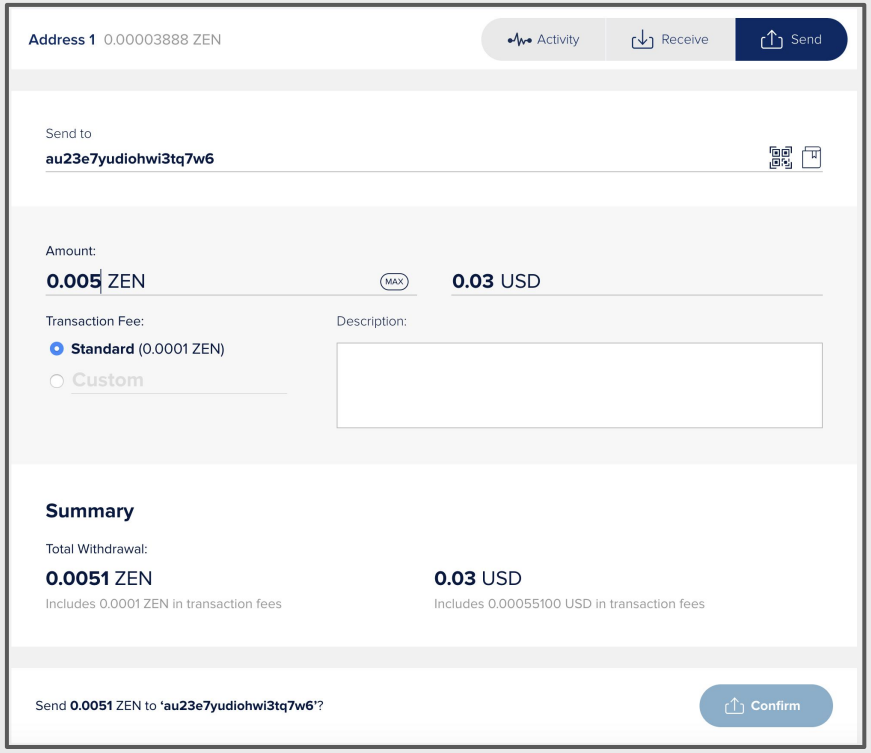
Click on the red Send Zen button to open up the Sending screen.



Transfer money to the game wallet

Fill in the address, and the amount in this screen and click on confirm.

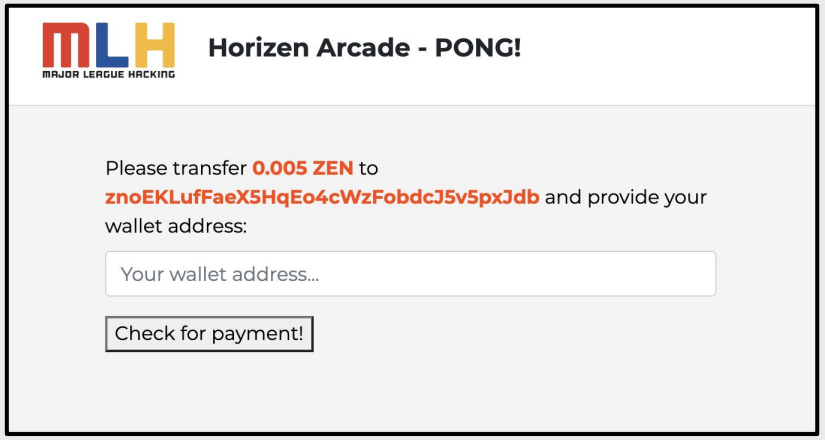
You will have to enter your wallet password to confirm.



Play the Game

Once the transaction is sent, type in your address into the text box of the application and click on “Check for Payment”.

You will be redirected to the game screen.

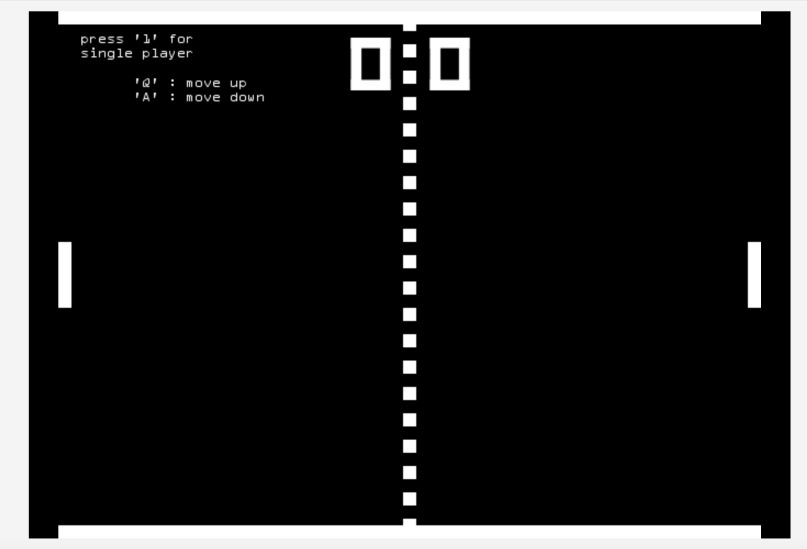


Play the Game

For single player, press “1” in your keyboard.

"Q" moves your paddle UP and "A" moves your paddle DOWN.

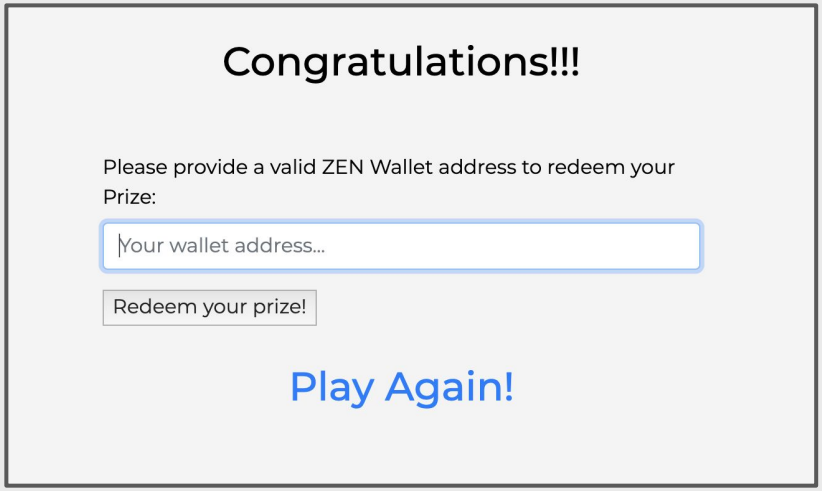
If you lose, try again!



Play the Game

If you won, you will need to input your wallet address to collect your winnings.

Enter the address of your Sphere wallet, and click on Redeem your prize!



Uh Oh!

You didn’t get any money in your wallet even though you were promised some.

Where do you think the issue is?

Getting money in your wallet

lib/zen/
index.js

```
57 const createTransaction = async (gameAddress, address) => {
58     // Retrieve full wallet
59     const addresses = getWalletAddresses();
60     const gameWallet = addresses.find(addr => addr.address === gameAddress);
61     const { privateKey } = gameWallet;

64 const blockHeight = 450150;
65 const bip115BlockHeight = blockHeight - 150;
66 const bip115BlockHash =
67    "0000000007844e62d471b966cc5926bd92e56a27d2c6a6ac8f90d34e11d3028d";

69   // Retrieve UTXO
71   // Create Raw Transaction
73   // Sign Transaction
75   // Serialize Transaction
77   // Broadcast Transaction
```

Open up the `index.js` file inside `lib/zen`

Do you see the unfinished createTransaction function? Let's fill it up to get money in your wallet!

First, let us learn a little bit more about transactions in the blockchain.

Transactions

When a transaction is sent on a blockchain, it is “locked” and can only be unlocked by the private key of the receiver.

Imagine that you received 10 ZEN from your grandmother as a Christmas gift. You now wish to spend 2 ZEN to buy a nice pair of headphones.

Let’s walk through what would happen.

Transactions

You would need to create a **transaction** in which you send the appropriate amount of money to the store to buy your pair of headphones.

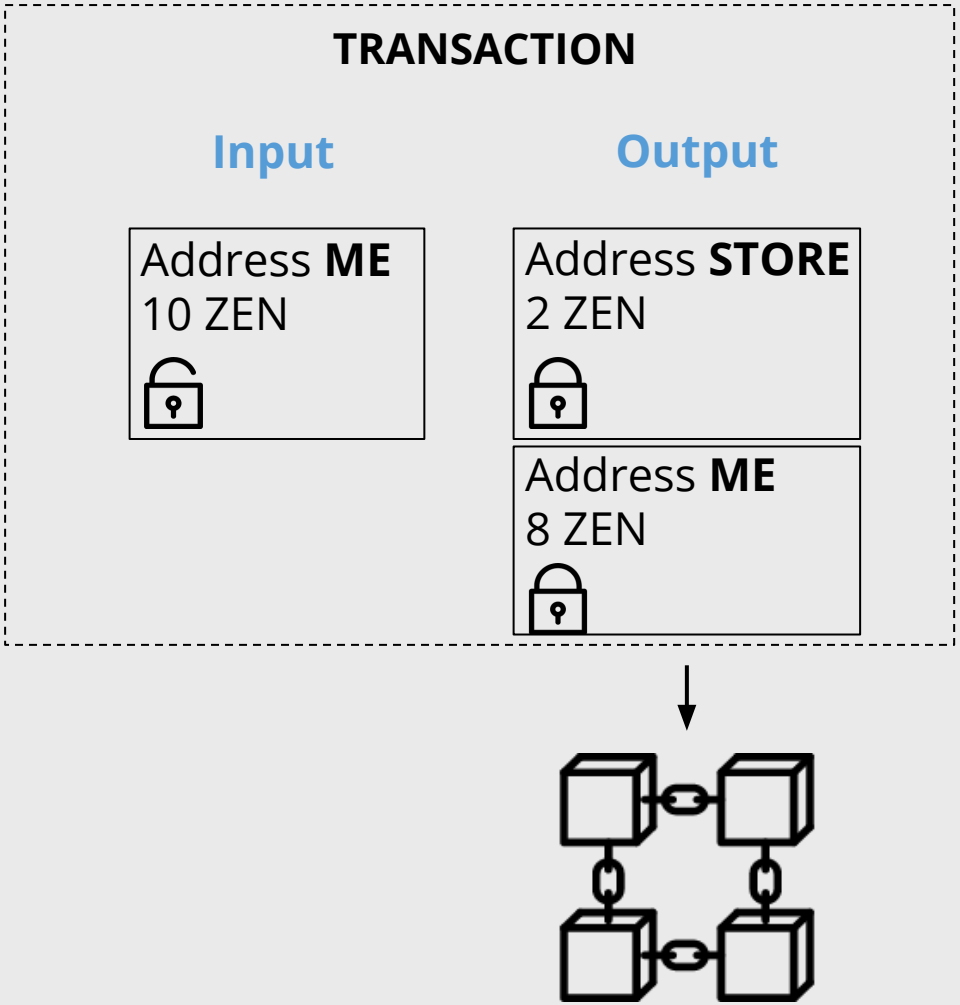
A transaction comprises **inputs** and **outputs**.

Inputs are the amounts spent by the sender.

Outputs are split into the following parts -

- the output that you want to send the payee(s),
- the change that gets added back to your wallet.

Transactions



The **input** to this transaction is the 10 ZEN your grandmother gave you.

The 10 ZEN gets split into 2 **outputs** -

- 2 ZEN for the store
- 8 ZEN change

Transactions

Remember when we mentioned that all transactions are locked?

To spend the 10 ZEN received with your public key, you have to unlock it using your **private key**.

The private key is used to create a **digital signature** which, if valid, unlocks your money.

Once the signed transaction is ready, we broadcast it to the blockchain.

Nodes and miners verify the sign is valid before including this transaction in the next block that is created.

Transactions

The outputs of the transaction are again, **locked**.

Locked transactions are also called **unspent transaction outputs** (or utxo).

The 10 ZEN that was the input to this transaction, is recognised as spent.

When you own X ZEN, that means -

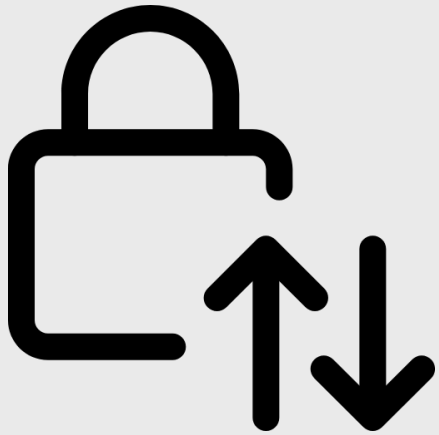
1. You received X ZEN as a transaction output, and
2. You haven't spent that output yet.

Steps to perform a transaction

RECAP

First, get all the required information -

1. Private key of sender.
2. Unspent transaction outputs.
3. Address of recipient.



Then perform the following steps -

1. Create the transaction
2. Sign the transaction
3. Serialize the transaction
4. Broadcast the transaction

`createTransaction` function

lib/zen/index.js

```
57 const createTransaction = async (gameAddress, address) => {
58     // Retrieve full wallet
59     const addresses = getWalletAddresses();
60     const gameWallet = addresses.find(addr => addr.address === gameAddress);
61     const { privateKey } = gameWallet;
```

First let us look at the code already present.

The create transaction method takes two inputs -

1. `gameAddress`: Address of the game's wallet.
2. `address`: Address that we want to send money to.

First, we get the private key associated with the game wallet, since we need this to sign the transaction.

`createTransaction` function

lib/zen/index.js

```
57 const createTransaction = async (gameAddress, address) => {
58     // Retrieve full wallet
59     const addresses = getWalletAddresses();
60     const gameWallet = addresses.find(addr => addr.address === gameAddress);
61     const { privateKey } = gameWallet;
```

`getWalletAddresses()` function gets a set of addresses that belong to the game's wallet.

In **line 60**, we find the address that maps to the current game wallet.

Line 61 gets the private key of this wallet into the variable `privateKey`.

`createTransaction` function

lib/zen/index.js

```
64 const blockHeight = 450150;
65 const bip115BlockHeight = blockHeight - 150;
66 const bip115BlockHash =
67     "0000000007844e62d471b966cc5926bd92e56a27d2c6a6ac8f90d34e11d3028d";
```

The next set of lines are all constants that we will use while creating our transaction.

blockHeight - The block at which you want to commit this transaction. You can look at **mlhlocal.host/ZENinsight** to find the latest block on Horizen.

bip115BlockHeight - A block that is 150 - 600 blocks older than the current block. This is used for replay prevention.

Quick detour!

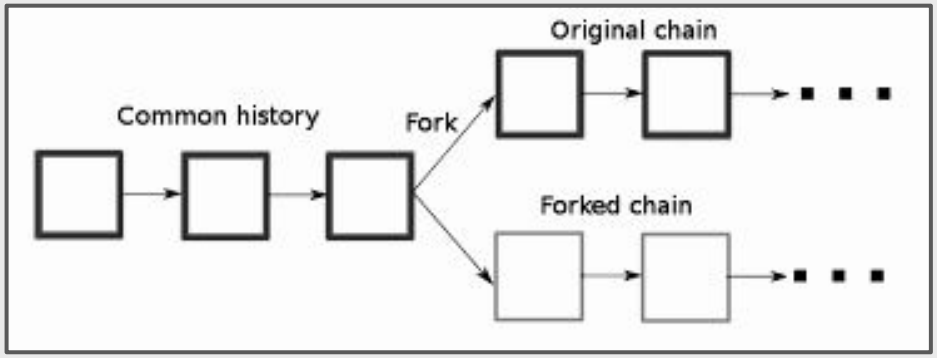
What is replay prevention?

Replay attack

A change in the code/ rules of a blockchain leads to a fork.

If some users reject the change, it causes a **hard fork**.

After a hard fork, if a transaction in one fork is replicated in another fork, it is called a **replay attack**.

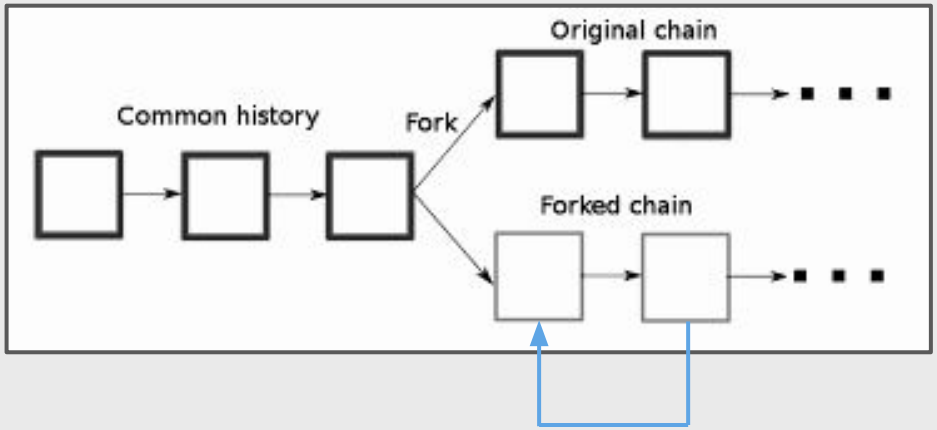


Replay prevention

To prevent replay attacks, we need to pin our transaction to a block that is -

- only on one of the forks, and
- will not be orphaned.

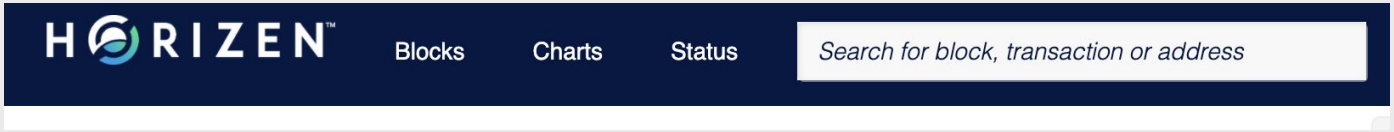
If the block used for the replay protection gets orphaned the transaction will be unspendable for at least 52596 blocks.



Back to the code.

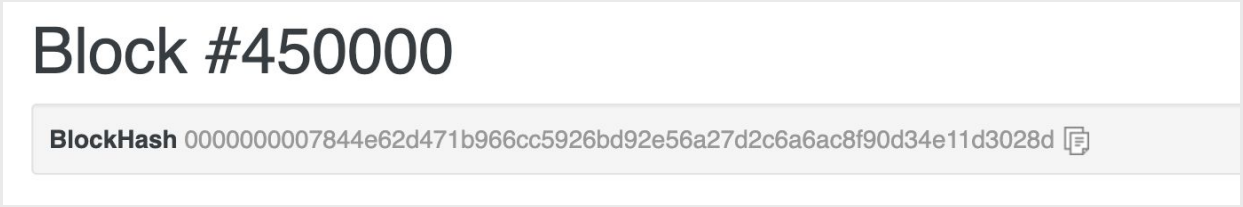
`createTransaction` function

To find the hash of a particular block, go to **mlhlocal.host/ZENinsight**, type the block number whose hash you need, and press enter.



In the page that pops up, you can see the blockhash for the block just below its number.

Here, you can see the hash for block #450000.



Your Turn - Complete `function createTransaction()`

lib/zen/index.js

```
57 const createTransaction = async (gameAddress, address) => {

69   // Retrieve UTXO
71   // Create Raw Transaction
73   // Sign Transaction
75   // Serialize Transaction
77   // Broadcast Transaction
```

What should this function do?

1. Get the transaction details.
2. Create the transaction.
3. Sign the transaction.
4. Serialize and broadcast the transaction.

Your Turn - Complete `function createTransaction()`

lib/zen/index.js

```
70 const utxo = await getUtxo(gameWallet.address);
71 const transactionDetails = [{ address, satoshis: utxo.satoshis }];
```

Let's get the unspent txn outputs for the game wallet's address.

We already have a function called `getUtxo` that gets this data for us. Let us call that. It requires one parameter - the address of the game wallet.

Now, we create an object that stores the transaction details -

1. The receiver's wallet address
2. The amount of unspent txn output in satoshis.

1 Get the transaction details.

Your Turn - Complete `function createTransaction()`

lib/zen/index.js

```
74 let rawTx = zencashjs.transaction.createRawTx(
75     [utxo],
76     transactionDetails,
77     bip115BlockHeight,
78     bip115BlockHash
79 );
```

Let's create the raw transaction by using the `createRawTx` method of the zencashjs library. It takes the following inputs -

1. Utxo
2. Transaction details
3. bip115BlockHeight
4. bip115BlockHash

2 Create the transaction.

Your Turn - Complete `function createTransaction()`

lib/zen/index.js

```
82 const signedTx = await zencashjs.transaction.signTx(
83     rawTx,
84     0,
85     privateKey,
86     true
87 );
```

Sign the transaction using the `signTx` method by supplying the following params -

1. The raw transaction
2. The index of the transaction in the raw transactions list
3. The private key of the game wallet address
4. Whether we should compress the public key or not

3 Sign the transaction.

Your Turn - Complete `function createTransaction()`

lib/zen/index.js

```
89 // Serialize Transaction
90 const serializedTx = zencashjs.transaction.serializeTx(signedTx);

92 // Broadcast Transaction
93 const broadcastedTx = await InsightApi.broadcastTransaction(serializedTx);

95 return broadcastedTx;
```

To serialize the signed transaction, we use `serializeTx` function.

And then broadcast it using the `broadcastTransaction` method that's already written in the insightApi.js file.

4 Serialize and broadcast the transaction.

And we’re done!

We should be able to get the money into our wallet now.

Play the game again and verify!

Play the Game

Play the Game

Hurray!!

Your prize has been sent!

Check the transaction here: https://explorer.horizen.global/tx/3e351d502adee6432a1dfb3195e1ebffb0b939ebe0dfb2e900867e71348cb223

Play Again!

Now all that's left to do is recap what we've learned

Let’s recap!

We launched the app and learned how to create and send a transaction to the blockchain to receive our prize money.

Horizen Early Adopter Program

mlhlocal.host/HEAP

Horizen Early Adopter Program (HEAP) is an opportunity for developers to gain early access to Horizen's newest releases and product features before they are publicly available. HEAP members also receive

- Invite-only webinars,
- pre-release product demos,
- tutorials and other free e-learning materials.

HEAP is currently accepting enrollments!

Horizen Developer Environment

mlhlocal.host/HDE

Contributing to an open-source project is easy with the Horizen Developer Environment (HDE)!

HDE is also a social platform and educational resource for all levels of developers. It provides developers with the opportunity to work on cutting edge technologies alongside the Horizen engineering team and developers from around the world.

Get FREE Horizen Swag!

Interested in some awesome swag from Horizen? Fill out this form and they will send you some awesome Hacker Gear!

mlhlocal.host/HorizenForm

Table of Contents

Let's Recap

What did you learn?

1. An Introduction to Blockchain technology and Horizen
2. How to create a Sphere wallet account to use ZEN coins
3. How to create and send a transaction to the Horizen blockchain.

Keep Learning: Practice Problems for Later

Extra Practice Problem 1:

Can you make this a 2-player game?

Extra Practice Problem 2:

If you were to create a sidechain specifically designed to support our Pong application, what would you change compared to the Horizen mainchain we used today?

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

http://mlhlocal.host/quiz

Learning shouldn't stop when the workshop ends...

Check your email for access to:

- These workshop slides
- Practice problems to keep learning
- Deeper dives into key topics
- Instructions to join the community
- More opportunities from MLH!

MLH localhost

Workshop

The Basics of Blockchain with Horizen

MLH localhost

HORIZEN™

v.0.0.1