RVConcerts - DaPP

FAQ

Frequently asked questions.

What are NFTs?

*NFTs (Non-Fungible Tokens) are unique **digital assets** that are secured and contained on the blockchain. The blockchain verifies ownership, provenance, and transaction history.*

What is Blockchain?

*Blockchain is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system. A blockchain is essentially a **digital ledger** of transactions that is duplicated and distributed across the entire network of computer systems on the blockchain.*

What is a “wallet”?

*Your crypto wallet is where you can view your cryptocurrency assets and transactions, all in one place. It’s also where you confirm any transactions in process. Different wallets have different features. For example, some allow you to view NFTs from a smartphone. Popular wallets include Coinbase and **Metamask.***

What is “minting”?

*NFTs are created through a process called minting. On the **Ethereum network**, a command is run that creates the NFT as an immutable record, providing it with a unique hash that contains all of the metadata. The metadata stores key contextual information about your NFT, like its current ownership and transaction history. Sites including opensea.io and etherscan.io allow you to view your NFT collection and NFTs in other collections.*

What is ‘Gas’?

The blockchain depends on processing power provided by a decentralized network of computers. “Gas fees” are what the members of this network earn in return for the processing power they provide. These fees can fluctuate according to the amount of traffic on the blockchain at any given moment.

What makes an NFT different from other kinds of digital art?

When you collect an NFT, it has a public record on the blockchain it was created on. This makes provenance public and verifiable, and ***permits for instantaneous transactions*** *on the secondary market.*

What are you doing about the environmental impact of your NFT collection?

Ryoshi Vision Concerts [RVC] - Events one-of-one NFTs will be made available on Ethereum, and the limited edition NFTs will be powered by Immutable X (minted and traded with zero gas fees).

RVC chose to partner with Immutable X, because all NFTs traded on Immutable X are completely carbon neutral.

How can I pay for my NFTs?

NFTs can be purchased with the cryptocurrency Ethereum (ETH). To purchase with ETH, first you'll have to get Ether (see full details). Once you have ETH, you will need to transfer it to your wallet, such as MetaMask. You can then connect your MetaMask wallet to make your purchase, confirm the transaction and then receive your NFT.

How can I get ETH (Ether)?

First, you must purchase Ethereum (ETH) through an online exchange such as Coinbase, Binance, etc. Once you have purchased ETH, you will need to transfer those funds into your Metamask or other compatible wallet in order to buy an NFT token. Please note there are limits on the amounts of ETH you can buy and sell based on your account and the platform you're using to purchase ETH. There are also time delays from when you buy your ETH to when you can transfer it to your wallet to make NFT purchases. Most platforms require you to hold the money in your wallet before you can transfer it to use for purchases.

What Can I Do with my NFTs?

NFTs can be collected, sold and traded on the blockchain. NFTs can also be tied to specific benefits, rewards, or experiences, depending on the program and particular details of each NFT.

How Can I Protect my NFTs?

Your digital wallet will come with a 12-24 word 'seed phrase' that acts as the password to your wallet. Never, ever share this seed phrase with anyone. We also recommend keeping a written, physical copy of your seed phrase for access offline. If another person gains access to your seed phrase they can take control of your wallet and everything it contains. If you lose access to your seed phrase, you effectively lose access to your wallet as well so keep it safe!

Where is my NFT collection?

You can view your NFTs directly in your wallet, either on a smartphone or desktop computer. You may use digital marketplaces that aggregate digital collectibles and NFTs under your ETH address; such as OpenSea, Rainbow, Lazy, Etherscan, and more.

How Long Until I Receive my NFT?

Depending on the amount of transactions pending on the blockchain, it may take up to 24 hours for the NFT to land in your wallet when you claim it. This is totally normal.

What is the Primary Market?

Primary Market or Primary Sales are the first time an NFT is sold. This can either be a purchase from the

NFT creator or an NFT that is minted on purchase.

What is the Secondary Market?

“Secondary market” refers to all NFT sales that occur after the initial NFT purchase. The most popular secondary marketplace to buy NFTs include OpenSea.io. As the owner of a Ryoshi Vision Concerts [RVC] - Events NFT, you can choose to either collect your NFT, or choose to sell it on marketplaces like Opensea and TokenTrove.

The Basics

1. Installation

1.1 Install Node

First you should install node in your system.

Windows: https://nodejs.org/en/download/

Linux: https://www.geeksforgeeks.org/installation-of-node-js-on-linux/

1.2 Install mongo DB

Second, you should install mongo DB server.

https://docs.mongodb.com/manual/installation/

2. About project

This explains that project is developed by what framework, template ,library.

The project is MERN stack web site.

This project is Dapp project integrated smart contract in Ethereum with web3, etherjs library.

This project uses express framework in node server.

This project uses React framework in frontend.

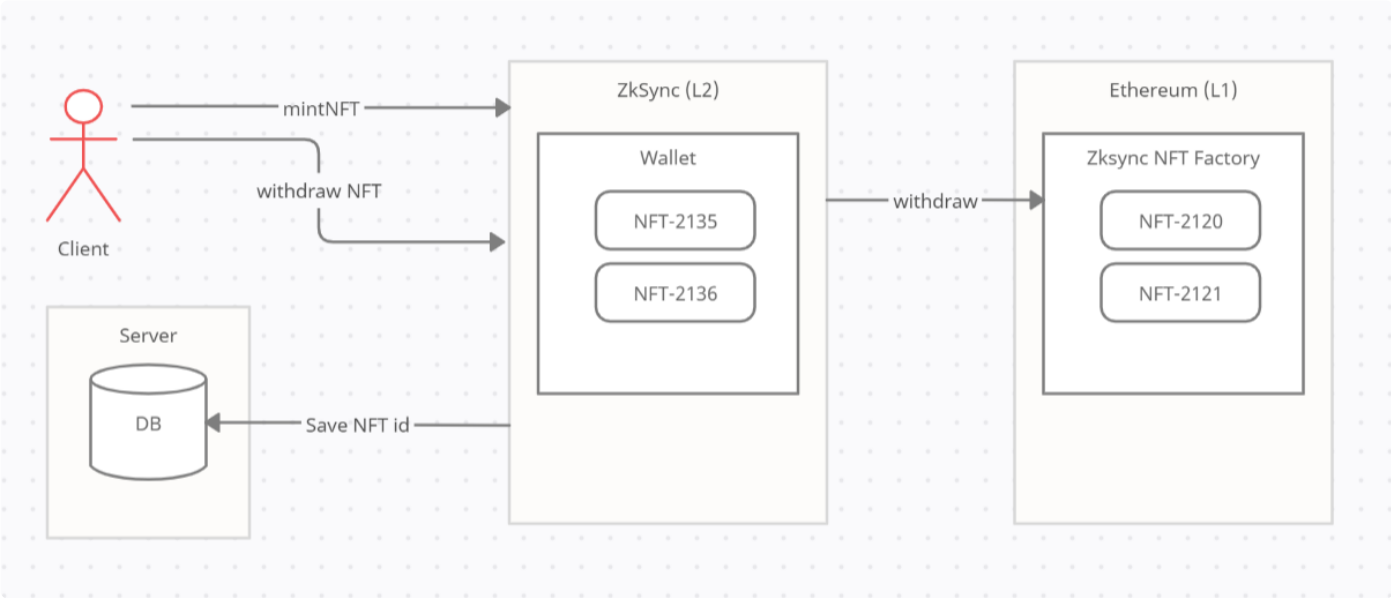
This project uses Mongo DB.

This project is working with zksync which is L2 solution of Ethereum.

This project mints NFT in zksync and withdraws it to L1(Ethereum mainnet).

This project uses Ryoshi ERC-20 token to give users who mint NFT.

Usecase



3. Source Directory

This explains the directory structure of project.

3.1 Client

This is a frontend with views, controller, reducer, route.

public

This has a index.html and logo.png and some CSS files.

src

This has frontend code.

actions: There are actions to control redux.

artifact: There are abi of compiled smart contract. The abi is for interact with smart contract deployed in blockchain.

components: There are react components displayed on web browser.
lib: There are codes to interact with smart contract.

reducers: There are reducer files.

utils: There are some utility functions used in special purpose in whole project.

3.2 config

There are config file about mongoDB.

3.3 models

There is mongo DB collection schema file.

3.4 routes

There are functions of server related to URL.

3.5 server.js

This file is to start node server and connect mongo DB.

4. Client side

This explains about frontend view and actions of redux.

4.1 actions

4.1.1 manager.js

There are actions to integrate with server.

setStatus() : set status text in page for test.

setAlert() : open Modal dialog and alert dialog.

setAccount():

check whether metamask installed.

```
if (window.ethereum) {
```

Get providers of zksync, ethereum by setting network and gets signer from provider to make wallet which can be used to sign transactions.

```
const syncProvider = await zksync.getDefaultProvider('rinkeby');
let provider;
let signer;
provider = new ethers.providers.Web3Provider(window.ethereum);
signer = provider.getSigner();
let syncWallet = await zksync.Wallet.fromEthSigner(signer, syncProvider);
```

And in case of accessing acount at first, it should be unlocked before accessing it.

```
      const res = await unlockAccount(syncWallet);
```

createNFT():

create NFT in manager UI with metadata Uri, email contents which Ryoshi gives to clients.

Post request to server to create new NFT with token Uri, amount, price, max amount(limit which can be minted in one wallet), email contents.

If creating success, get NFT's data from metadata and save it into store.

```
const res = await api.post('/ryoshi/create', { tokenUri, amount, price, maxAmount, emailCo
    nft = res.data;

    const content = await axios.get(nft.tokenUri);
    nft = { ...nft, ...content.data };
    dispatch(setAlert(true, `${amount} ${nft.name} NFTs are created`));
    dispatch({
      type: CREATE_SUCCESS,
      payload: nft
    });
```

updateNFT():

update NFT by sending request to sever with updated NFT's data.

```
await api.post('/ryoshi/update', { id, amount, price, maxAmount, emailContents });
```

getNFTs():

get NFTs from server. Type specifies whether NFTs are in drops, market, manager page.

Type can be 0: all, 1: market, 2: drop.

```
const res = await api.post('/ryoshi/nfts', { type });
let fullNfts = [];
let content;
...
for (let i = 0; i < res.data.length; i++) {
      await axios.get(res.data[i].tokenUri).then((response) => {
            content = response.data;
      });
      fullNfts.push({ ...res.data[i], ...content });
}
```

getAssets():

get NFTs which are bought with fiat by inputting email and key.

```
const res = await api.post('/ryoshi/assets', { email, account });
```

getZkNFTs():

Get NFTs from zksync.

Get commited and verifed NFTs by zksync wallet.

```
const state = await zksyncWallet.getAccountState(zksyncWallet.address);
const comitNfts = state.committed.nfts;
const comitNftIds = Object.keys(comitNfts);
```

And get token Uri from content hash and get NFT's data by metadata and get price and email contents from server.

```
for (let i = 0; i < comitNftIds.length; i++) {
  let content;
  await axios.get(TOKEN_URI + getCIDFromContenthash(comitNfts[comitNftIds[i]].contentHash)
    content = response.data;
  });
  const res = await api.post('/ryoshi/nftdata', { id: comitNfts[comitNftIds[i]].id });
  fullNfts.push({ id: comitNfts[comitNftIds[i]].id, symbol: comitNfts[comitNftIds[i]].symb
}
```

getWithdrawNFTs():

Get NFTs from Ethereum(L1) with wallet address and zksync factory address.

```
const zk_res = await axios.get(`https://deep-index.moralis.io/api/v2/${zksyncWallet.addres
```

In the above, it uses Moralis API which is used to get data from blockchain database.

buy():

mint NFT with crypto or start to buy with fiat.

First it check the balance of ryoshi and if client has enough ryoshi to mint NFT, it allows minting to client.

```
const balance = await getTokenBalance(account, zksyncWallet);
if (balance >= RYOSHI_PER_NFT * amount) {
  dispatch(setStatus(`${amount} NFTs(${nft.name}) is minting now...`));
  dispatch(setMintingNow(true));
  let receipt = await mintWiCry(zksyncWallet, amount, nft.tokenUri, nft.minPrice);
```

mintWiCry function does minting in zksync. This is explained in detail at lib section.

If client doesn't have enough ryoshi, it shows payment dialog to buy with fiat.

```
dispatch(openPay(true));
```

checkMaxOwnership():

check whether the ownership amount is exceed the limit.

It sends request for checking ownship limit to server and get result.

```
await api.post('/ryoshi/maxlimit', { account, email, id, amount });
```

approve():

buy NFT with fiat, however NFT is just stored in DB with approved status as a specific email.

NFT is approved to client with email and amount to mint later.

```
await api.post('/ryoshi/approve', { owner: email, id: nft._id, amount });
```

mint():

mint NFT which was approved by buying with fiat.

Before minting, it checks max ownership.

```
const res = await checkMaxOwnership(account, "", nft._id, amount);
```

If max ownership checking is passed, it checks whether key is valid with NFT.

If email is valid, then it mints NFT in zksync and transfer ryoshi related to amount of minting from NFT creator to client.

```

```

```
  const email = await checkKey(key, nft);
  if (email) {
    try {
      let receipt = await mintWiFia(zksyncWallet, amount, nft.tokenUri);
      if (receipt) {
        const transferRes = await api.post('ryoshi/transferRyoshi', { amount, to: zksyncWalle
```

And minting is succeeded, it updates NFT's status to minted and add new minted NFT ids in zksync by sending request to server.

```
await api.post('ryoshi/mint', {
    id: nft._id,
    owner: account,
    amount,
    email,
    nftIds: receipt
});
```

withdrawNFTtoL1():

withdraw NFT to Ethereum L1 by NFT id in zksync.

withdrawNFT function is in lib folder. We will discuss about later in lib section.

```
const res = await withdrawNFT(zksyncWallet, nftId);
```

setPay(): open payment dialog

setBuyNftAmount(): set NFT and its amount which is going to buy

checkKey():

get email which has a key and approved specific NFT.

```
const res = await api.post('/ryoshi/check', { key, id: nft._id });
```

openInputKey(): open dialog which input key

deleteNft():

delete NFT in manager page by manager.

First, it checks whether client is manager.

```
if (!CREATORS.find(item => item.toLowerCase() === account)) {
```

And delete NFT from DB by sending request to server.

```
const res = await api.delete(`/ryoshi/delete/${nft._id}`);
```

isManager(): check whether account is a manager

setShowNft():

in manager UI, set visibility of NFTs whether it is for drop, market, manager.

```
await api.post('/ryoshi/show', { id: nft._id, show });
```

4.1.2 types.js

This defines action types related to actions.

```
export const SET_STATUS = 'SET_STATUS';    // set status message in test mode
export const SET_ALERT = 'SET_ALERT';    // open popup dialog
export const SET_ACCOUNT = 'SET_ACCOUNT';    // set account with creating zksync wallet, u
export const OPEN_ZKWALLET_DLG = 'OPEN_ZKWALLET_DLG';    // open zksync wallet dialog
export const OPEN_SWAP_DLG = 'OPEN_SWAP_DLG';    //    open swap dialog
export const SET_UNLOCK = 'SET_UNLOCK';    // set unlock status of account
export const SET_ZKSYNCWALLET = 'SET_ZKSYNCWALLET';    // set zksync wallet
export const CREATE_NFT = 'CREATE_NFT';    // create NFT
export const CREATE_SUCCESS = 'CREATE_SUCCESS';
export const UPDATE_SUCCESS = 'UPDATE_SUCCESS';
export const LOADING_ASSETS = 'LOADING_ASSETS';
export const GET_NFTS = 'GET_NFTS';    // get NFTs to be displayed in market and manager pa
export const GET_ZK_NFTS = 'GET_ZK_NFTS';    // get NFTs in zksync
export const GET_WITHDRAW_NFTS = 'GET_WITHDRAW_NFTS';    // get NFTs in mainnet
export const GET_NFT_CONTENT = 'GET_NFT_CONTENT';    // get NFTs' data by metadata
export const MINT_SUCCESS = "MINT_SUCCESS";
export const WITHDRAW_NFT = 'WITHDRAW_NFT'    // withdraw NFT to L1
export const SET_PAY = 'SET_PAY';    // set payment result
export const SET_BUY_NFT_AMOUNT = 'SET_BUY_NFT_AMOUNT';    // set NFT and amount to buy
export const OPEN_PAY = 'OPEN_PAY';    // open payment dialog
export const APPROVE_SUCCESS = 'APPROVE_SUCCESS';
export const OPEN_INPUT_KEY = 'OPEN_INPUT_KEY';    // open key input dialog
export const DELETE_SUCCESS = 'DELETE_SUCCESS';
export const SET_MINTING_NOW = 'SET_MINTING_NOW';    // set if it is minting now
```

4.2 artifacts

The abi.js is abi of smart contract to interact with deployed contract.

```
export const ryoshi_token_abi = [
        {
                "inputs": [
                        {
                                "internalType": "address",
                                "name": "tokenOwner",
```

```
                                "type": "address"
                        }
                ],
                "name": "balanceOf",
                "outputs": [
                        {
                                "internalType": "uint256",
                                "name": "",
                                "type": "uint256"
                        }
                ],
                "stateMutability": "view",
                "type": "function"
        },
        {
                "inputs": [],
                "name": "decimals",
                "outputs": [
                        {
                                "internalType": "uint8",
                                "name": "",
                                "type": "uint8"
                        }
                ],
                "stateMutability": "view",
                "type": "function"
        }
];
```

4.3 components

This part have react components with html. And it use actions defined in actions/manager.js.

Actions are already explained above so in this part it doesn't explain code.

4.3.1 client

assets: This contains NFT view components in asset page

drops: This contains NFT view components in drop page

nfts: This contains NFT view components in market page

Assets.js: This is a asset page component.

Drop.js: This is a drop page component.

Market.js: This is a market page component.

styles.css: youtube embed component's style

youtubeEmbed.js: youtube embed player component

4.3.2 dialog

Alert.js: modal dialog component

CheckoutForm.js: stripe card component

InputKey.js: inputting key component

Payment.js: stripe payment component

4.3.3 layout

Footer.js: footer component

Navbar.js: navigation bar component

NavDrawer.js: in mobile, left side navigation bar

particleConfig.json: Particle effect configuration in navigation bar

4.3.4 manager

nfts: nft view component in manager page

manager.js: manager page component

4.3.5 routing

Routes.js: this defines routing component

4.3.6 StyledComponent

There are customized material UI components.

4.4 lib

This includes functions interact with blockchain.

4.4.1 minter.js

defines address constants.

```
1 const RYOSHI_ADDRESS_TEST = "0x5290f2a89654d5fe181858929c931fce17550a77";
2 const DAI_ADDRESS_TEST = "0x2e055eee18284513b993db7568a592679ab13188";
3 const CREATOR = "0xCe9499b23a087d2494956C33a064E075EC23dafc";
```

transferEth():
transfer ETH to account in zksync.

```
const transferTransaction = await wallet.syncTransfer({
    to,
    token: "0x0000000000000000000000000000000000000000", // ETH address
    amount: ethers.utils.parseEther(price.toString()),
    // fee: ethers.utils.parseEther("0.001")
});

// Wait till transaction is committed
const transactionReceipt = await transferTransaction.awaitReceipt();
```

to: account address which transfer ETH to.

token: token address, (0x0000000000000000000000000000000000000000: ETH address)

amount: token amount which transfer. ethers.utils.parseEther function convert value to ethereum decimals(18decimals)

awaitReceipt function waits until transaction is receipted by zksync.

mintWiCry():

mint NFT with Eth in zksync.

First it makes content hash of token Uri of metadata to store in zksync.

Zksync mints NFT with content hash of metadata Uri.

```
const contentHash = "0x" + getContentHashFromUri(tokenUri);
```

Before minting NFT, it transfer ETH for NFT price to creator.

```
const res = await transferEth(zksyncWallet, CREATOR, minPrice * amount);
```

And create minting NFT transaction with receiver address and content hash.

Fee of transaction is set to ETH.

```
const nft = await zksyncWallet.mintNFT({
    recipient: zksyncWallet.address(),
    contentHash,
    feeToken: "ETH"
})
const receipt = await nft.awaitReceipt();
```

And get all commited and verifed NFTs of zksync address to get new committed NFTs

Commited NFT is one on pending of minting.

Verified NFT is one on verified already-finish pending.

```
const state = await zksyncWallet.getAccountState(zksyncWallet.address());
const comitNftIds = Object.keys(state.committed.nfts);
mintedNFTs = comitNftIds.slice(comitNftIds.length - amount);
```

mintWiFia():

mint NFT without transfering ETH to creator, after that, it get Ryoshi from creator.

this function is same as mintWiCry function except that transfering ETH.

withdrawNFT():

withdraw NFT to Ehtereum L1 by NFT id in zksync.

Create withdrawNFT transaction with receiver address, NFT id, fee token(default "ETH").

```
const withdraw = await zksyncWallet.withdrawNFT({
    to: zksyncWallet.address(),
    token: nftId,
    feeToken: "ETH"
});
const receipt = await withdraw.awaitReceipt();
```

unlockAccount():

It need to unlock account to access account in zksync af first.

This means that sets signing key to account in zksync.

```
if (!(await syncWallet.isSigningKeySet())) {
        if ((await syncWallet.getAccountId()) === undefined) {
            throw new Error("Unknown account");
            return false;
        }

        try {
            // As any other kind of transaction, `ChangePubKey` transaction requires fee.
            // User doesn't have (but can) to specify the fee amount. If omitted, library w
            // the lowest possible amount.
            const changePubkey = await syncWallet.setSigningKey({
                feeToken: "ETH",
                ethAuthType: "ECDSA",
            });

            // Wait until the tx is committed
            const changeReceipt = await changePubkey.awaitReceipt();
```

getTokenBalance():
Get total ryoshi balance of account in mainnet and zksync.

```
//    get balance in mainnet
let ryoshiContract = new window.web3.eth.Contract(ryoshi_token_abi, RYOSHI_ADDRESS_TEST);
let ryoshi_L1 = ethers.utils.formatEther(await ryoshiContract.methods.balanceOf(account).ca

//    get balance in zksync
let ryoshi_L2 = 0;
if (zksyncWallet)
    ryoshi_L2 = ethers.utils.formatEther(await zksyncWallet.getBalance(RYOSHI_ADDRESS_TEST,
return Number(ryoshi_L1) + Number(ryoshi_L2);
```

to_b58(): converts uint8Array to b58 string.

from_b58(): converts b58 string to uint8Array.

fromHexString(): converts hex string to uint8array.

toHexString(): converts uint8array to hex string.

getContentHashFromUri():

get content hash from metadata Uri. It remove first 4 charaters ("1220") to make length of hash to 64.

```
let cid = tokenUri.split('/')[tokenUri.split('/').length - 1];
var MAP = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
var decoded = toHexString(from_b58(cid, MAP));
return decoded.slice(4);
```

getCIDFromContenthash():

get CID to be metadata Uri from content hash.

```
var MAP = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
hash = '1220' + hash.slice(2);
return to_b58(fromHexString(hash), MAP);
```

4.5 reducers

This defines reducer for actions.

This sets states of store every incoming

4.6 utils

4.6.1 api.js

This create axios with base URL, headers.

```
const api = axios.create({
  baseURL: 'http://localhost:5000/api',
  headers: {
    'Content-Type': 'application/json'
  }
});
```

4.6.2 utils.js

checkEmail(): check whether email is valid type.

isBigger(): compare two string of number

shortAddress(): shorten wallet address

isInt(): check whether n is integer

4.7 app.js

This defines root component.

5. Server side

This explains server side script.

5.1 server.js

This connects mongo DB.

```
connectDB();
```

This connects APIs to controller function.

```
app.use('/api/ryoshi', require('./routes/api/ryoshi'));
app.use('/api/stripe', require('./routes/api/stripe'));
```

This starts server on port 5000.

```
app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

5.2 routes/api

This defines functions corresponding to API. These functions interact with mongo DB.

5.2.1 ryoshi.js

credentials:

defines mail sender and password and mail server.

```
const credentials = {
    service: 'GMail',
    auth: {
        user: 'guruluckystacker@gmail.com',
        pass: '*******'
    }
}
```

defines Ryoshi amount which transfered to client buying with fiat and Ryoshi token address.

```
const RYOSHI_PER_NFT = 0.01;    // TEST
const RYOSHI_ADDRESS_TEST = "0x..............";
```

router.post('/create'):

Add NFT into DB with token Uri, price, amount, ownership limit, email contents.

```
 const newNft = new Nft({
  tokenUri: tokenUri,
  minPrice: price,
  approveOwners: [],
  mintOwners: [],
  amount: amount,
  maxAmount,
  left: amount,
  emailContents
});
const nft = await newNft.save();
```

router.post('/update'):

update NFT in DB with updated amount, price, limit, left amount, email contents.

```
const result = await Nft.updateOne({ _id: id }, { $set: { amount, minPrice, maxAmount, left
```

router.post('/nfts'):

get NFTs from DB as its visibility which means drop, market, manager.

```
let type = req.body.type;
let nfts;
switch (type) {
    case 1: //market
        nfts = await Nft.find({ show: { $in: [0, 1] } }, { tokenUri: 1, minPrice: 1, amount
        break;
    case 2: //drop
        nfts = await Nft.find({ show: { $in: [0, 2] } }, { tokenUri: 1, minPrice: 1, amount
        break;
    case 0: //all
    default:
        nfts = await Nft.find({}, { tokenUri: 1, minPrice: 1, amount: 1, maxAmount: 1, left
        break;
};
res.json(nfts);
```

router.post('/assests'):

get NFTs from DB corresponding to email and key.

```
et email = req.body.email.split("+")[0] || "";
    let key = req.body.email.split("+")[1] || "";
    let account = req.body.account || "";
    let nfts = [], approves = [], mints = [];
    let approveNfts, mintNfts;
    if (email) {
        approveNfts = await Nft.find({ "approveOwners.address": email, "approveOwners.key"
        approves = approveNfts.map(nft => {
            let amount = nft.approveOwners.find(owner => owner.address === email).amount;
            let ownerId = nft.approveOwners.find(owner => owner.address === email)._id;
            return {
                _id: nft._id,
                ownerId,
                tokenUri: nft.tokenUri,
                minPrice: nft.minPrice,
                amount: amount,
                type: "approve"
            };

        });
    }
```

router.post('/nftdata'):

Get token Uri and price, email contents of minted NFT by NFT id.

```
let nftData = await Nft.find({ nftIds: Number(nftId) }, { minPrice: 1, emailContents: 1, t
```

router.post('/mint'):

update NFT's status 'mint' and add minteOwners document with minter address.

And it adds NFT ids minted in zksync.

It deletes minted NFTs from approved NFTs and adds them into minted NFTs.

```
let id = req.body.id;
let address = req.body.owner;
let amount = req.body.amount;
let email = req.body.email;
let nftIds = req.body.nftIds;
const nft = await Nft.findById(id);
let updatedMintOwners = [], updatedApproveOwners = [];
////////    removes minted NFTs from approveOwners
if (email) {
    if (nft.approveOwners.find(owner => owner.address === email)) {
        if (nft.approveOwners.find(owner => owner.address === email).amount === amount) {
            updatedApproveOwners = nft.approveOwners.filter(owner => owner.address !== ema
        } else {
            updatedApproveOwners = nft.approveOwners.map(owner => {
                if (owner.address === email) {
                    owner.amount -= Number(amount);
                    if (owner.amount == 0)
                        return;
                    else
                        return owner;
                }
                return owner;
            });
        }
    }
}
/////    Add minted NFT to mintOwners
if (nft.mintOwners.find(owner => owner.address === address)) {
    updatedMintOwners = nft.mintOwners.map(owner => {
        if (owner.address === address) {
            owner.amount += Number(amount);
            return owner;
        }
        return owner;

    });
} else {
    updatedMintOwners = [...nft.mintOwners, { address, amount }];
}
////////        Adds minted NFT ids in zksync
```

```
let result;
await Nft.updateOne({ _id: id }, { $set: { nftIds: [...nft.nftIds, ...nftIds] } });
if (email) {
    result = await Nft.updateOne({ _id: id }, { $set: { approveOwners: updatedApproveOwners
} else {
    result = await Nft.updateOne({ _id: id }, { $set: { approveOwners: updatedApproveOwners
}
```

router.post('/approve'):

add approveOwners document with approved email.

```
let updatedOwners = [];
if (nft.approveOwners.find(owner => owner.address === address)) {
    updatedOwners = nft.approveOwners.map(owner => {
        if (owner.address === address) {
            // console.log("exist", address);
            owner.amount += Number(amount);
            owner.key = key
            return owner;
        }
        return owner;
    });
} else {
    updatedOwners = [...nft.approveOwners, { address, amount, key }];
}
const result = await Nft.updateOne({ _id: id }, { $set: { approveOwners: updatedOwners, le
```

And it send Email with email contents to client who is approved to mint later.

```
const email = {
    from: 'guruluckystacker@gmail.com',
    to: req.body.owner,
    subject: 'Ryoshi vision',
    text: `${nft.emailContents.toString()} \n This is a key for minting all NFTs(${content
}

try {
    await transporter.sendMail(email);
    console.log(email);
} catch (err) {
    console.log('send mail', err);
}
```

router.post('/maxlimit'):

check whether the ownership amount of user exceeds limit.

check amount of NFTs approved by email.

```
if (account) {
    const mintOwner = nft.mintOwners.find(owner => owner.address === account);
    let minted = 0;
    if (mintOwner) {
        minted = mintOwner.amount;
    }
    if (minted + Number(amount) <= nft.maxAmount || nft.maxAmount == undefined) {
        console.log(`${account} maxlimit passed`);
        res.json({ checked: true });
    } else {
        console.log(`${account} maxlimit failed`);
        res.json({ checked: false, msg: `You can't mint more than ${nft.maxAmount} Nfts. Yo
    }
}
```

check amount of NFTs minted.

```
if (email) {
    const approveOwner = nft.approveOwners.find(owner => owner.address === email);
    let approved = 0;
    if (approveOwner) {
        approved = approveOwner.amount;
    }
    if (approved + Number(amount) <= nft.maxAmount || nft.maxAmount == undefined) {
        console.log(`${email} maxlimit passed`);
        res.json({ checked: true });
    } else {
        console.log(`${email} maxlimit failed`);
        res.json({ checked: false, msg: `You can't buy more than ${nft.maxAmount} Nfts. You
    }
}
```

router.post('/check'):

check whether the key inputted is valid and return NFT with that key.

```
const nft = await Nft.findOne({ "approveOwners.key": key, _id: id }).exec();
if (nft) {
    const content = await axios.get(nft.tokenUri);
    const email = nft.approveOwners.find(owner => owner.key === key).address;
    res.json({ email });
} else {
    res.josn({ email: null });
}
```

router.post('/delete/:id'):

delete NFT from DB

```
const result = await Nft.remove({ _id: id });
```

router.post('/show'):

set NFT's visibility to be displayed in drop, market, manager page.

```
const result = await Nft.updateOne({ _id: id }, { $set: { show } });
```

router.post('/transferRyoshi'):

transfer ryoshi to client who buy with fiat while minting.

First, create creator's zksync wallet with his private key.

```
const syncProvider = await zksync.getDefaultProvider('rinkeby');
const ethersProvider = await ethers.getDefaultProvider(
    'rinkeby',
    {
        alchemy: "https://eth-rinkeby.alchemyapi.io/v2/0b9-h6iQGcN5mNxCTdJSzByB5WLkqR5I"
    }
);

// const ethWallet = ethers.Wallet.fromMnemonic('taxi erode orbit enforce apology present
const ethWallet = new ethers.Wallet('***PRIVATE_KEY****').connect(ethersProvider);//creato
const syncWallet = await zksync.Wallet.fromEthSigner(ethWallet, syncProvider);
```

and creates syncTransfer transaction to transfer Ryoshi in zksync as amount of NFT minted.

```
const transferTransaction = await syncWallet.syncTransfer({
    to,
    token: RYOSHI_ADDRESS_TEST, // DAI address
    amount: ethers.utils.parseEther((RYOSHI_PER_NFT * amount).toString()),
});

// Wait till transaction is committed
const transactionReceipt = await transferTransaction.awaitReceipt();
```

5.2.2 stripe.js

create stripe object with stripe account's private key.

```
const stripe = require("stripe")("sk_test_51JXxg2CDjQLWm9hRmboRlONQctPfntwi9shPzFFk1K84483
```

router.post('/create-payment-intent'):

create stripe paymentIntents and returns payment client's secret.

```
router.post("/create-payment-intent", async (req, res) => {

```

```
    const { items } = req.body;
    // Create a PaymentIntent with the order amount and currency
    try{
        const paymentIntent = await stripe.paymentIntents.create({
            amount: 300,//cent 3$
            currency: "usd"
        });

        res.json({
            clientSecret: paymentIntent.client_secret
        });
    }catch(err){
        console.log(err);
    }

});

```

Database

1. Nft

This collection defines NFTs in DB

The NftSchema is following:

tokenUri: String, defines tokenUri of NFT (metadata URL)

minPrice: Numbr, defines price of NFT

approveOwners: [approveOwner], defines owners(email) who is approved this NFT and his key for minting later.

mintOwners:[mintOwner], defines owners(address) who mints this NFT.

nftIds:[Number], defines NFT ids to be minted in zksync.

amount: Number, defines NFT total supply.

left: Number, defines left amount of NFT.

emailContents: [], defines email contents of NFT

maxAmount: Number, defines the ownership limit of NFT.

show: Number, defines the visibility of NFT. 0: all show, 1: market, 2: drop, 3: all hide

Smart contract

Smart contract

There isn't any own smart contract.

We use zksync factory contract in mainnet to mint NFT which is provided by Zksync.

So we don't have own smart contract.

Test instruction

Settings before test

All setting are stored in both .env file.

Frontend setting is client/.env, Backend setting is .env in root directory which has a server.js file.

Frontend

1. REACT_*APP_NETWORK: ethereum network*
2. REACT_APP_CREATOR: creator's wallet address
3. REACT_*APPRYOSHI_PER_NFT: amount of ryoshi to be needed for buying multiple NFTs with crypto.*
4. *others are ryoshi, zksync factory contract address in mainnet, rinkeby. Don't need to change them.*

Backend

1. CREATOR_PK: creator's private key, creator's address is already set in frontend .env file.
2. EMAIL: email address which sends email contents.
3. PASSWORD: password of sender email.
4. STRIPE_USD: usd for buying with fiat. this value is cent.

Run server

If you change .env in frontend.

`cd ryoshi/client`

`npm run build`

Run server.
`sudo pm2 restart 0`

---

1. Manger

1. deposit ETH to creator account in zksync.
2. deposit Ryoshi to creator account in zksync.https://wallet.zksync.io/transaction/deposit
3. create NFTs. set lowest price for test Be careful min price, this is real ETH price.
4. check visibility for show all, only market, only drop...
5. check delete

---

2. Mercatus

1. as creator, deposit ETH enough for NFT price and gas fee and unlock client.
2. Don't deposit Ryoshi to client. Check buy with crypto without ryoshi.
3. If client buys with crypto without ryoshi or enough ryoshi for multiple minting, it show open payment dialog and buy with fiat.
4. buy with fiat(test stripe). And check receiving email with key & email contents. Be careful amount and price before buying ,however, NFTs aren't minted yet.
5. After minting in forge/pending to mint, check buy with crypto. check minting amount and ryoshi amount. If client has more than (1000*amount of NFT)Ryoshi, he c cheint multiple NFT at once. client pay exact ETH and creator gets exact ETH. Be careful amount and price before buying.

---

3. Forge

3.1 Pending to mint

1. mint NFT and check getting exact ryoshi. It will be 1000*amount of buying NFT. Be careful amount and price before minting.

3.2 Already minted

1. check Ryoshi ID. it starts from 1.
2. withdraw to L1. check gas price. it takes more gas fee than others in zksync.

3.3 Withdrawn

1. check NFT in etherscan.
2. check NFT in opensea.

---

4. Drop and Buy Ryoshi

there isn't special testing in these.