HYPER WIKI

OverView

The best way to think of the hyper wiki is a public multi-user server with limited scope, all accessed and used through the frontend API.

Users can store data in memory and get it back or change it to ake “Barter Tokens”, a simple cryptocurrency that does not require a blockchain or proof of work but works through a one-way hash. These barter tokens can also be trained and can be used to make posts private.

To use this system, users can post HTML pages that can access these features but can also be used and accessed by other languages like python.

Post and GET Data

Posting data can be done with or without a user. Once the **POST** command the data is published, you will get a memory address you can use to get the data. The GET command can use this data return.

Some aspects of the data are multistable and can be changed with a **Change post** command.

Barter Tokens

Barter tokens are a simple one-way hash.

To use the first, make a ledger with the **Add Ledger** command.

Once that’s is done, barter tokens from that ledger can be made; this is done with the **Add Key** command.

Once a key is made, three main things are stored a key-id, a random string of data to identify the key (*think of it as a username*). And a hash is a 256-bit string of information, the hash of a randomly generated 256-bit string of information that is returned to the user as a solution (*think of it as a password*). It will also store the name of the ledger and some secrete information from the key creator.

The return value is the key-id and the solution.
Form here, the holder of the secrete has three options.

They can cash the key with a key and receive a use key to verify that the key was cashed. This allows them to gain access to gated posts. It can also be used to send super chats with movement on the backend system.

They can change keys, allowing them to make a new key and destroy the old one. They pass the key-id and the solution along with the new hash. The old key is cashed (IE its solution is saved and made invalid), and a new key id is created with all the same information as the old key except the hash solution was now produced by the user. This makes it so the user can know that the key solution cannot be held by anyone else.

Or lastly, from could use them in the market.

Market

When you make an account, 1000 money1 and 1000 money2 are added to your account. You can see these when you use PRINT user. These are there just so you have something to trade, and they have no real value.

You can also use **Add Crypto** along with the ledger name KeyID and password to add a barter token to your account. You can also add one from other hyper wikis by stating a path.

You can get the crypto back with the **Get Key Back** command.

Next, you can **Make a triad**. You tell it the money you want the money are willing to give, and along with your user name and password if you are successful, a triad id will be created. You can with **Compleat triad** this triad id you can exchange information with another user or yourself.

Templates

Templates are uploadable HTML pages that use the only post command in steam.

Once a template is created, it can be viewed with the **make page** command in your browser.

When you're making your make page, you have the ability to use sessions and

JS commands

Pase Json

```
val = JSON.parse(“return statemant”);
```

Make command

```
function post_responce(path,func,varible){
  fetch(path).then(
            ( response) => {
            return response.text();
            })
            .then((html) => {
            func(  html.trim()  , varible )
            });
}

post_responce("comand",funtion,"var");
//post_responce("http://localhost:8000/doit?key=Keyid&action_type=get_post",this,"");



function replace(A,B){
  val = JSON.parse(A);
  console.log(val);
  alert(A);
}
```

Make template

```
var xhr = new XMLHttpRequest();

xhr.open("POST", "http://localhost:8000/doit", true);

//path xhr.open("POST", "http://localhost:8000/doit", true);

 xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

post_val ="user=User&password=password&temmplate_name=TemplateName&template="\
+encodeURI(val)+"&type=this&replace=!"

xhr.send(post_val);
```

Python Commands; this e json:

```
import json
x =  '{ "id": "POST_ID", "username": "u1", "text": "text", "body": "body", "tital": "", "time": "2021-10-09 00:43:40", "photo": "photo", "iframe": "", "catagoy": "cat", "catagoy_2": "" }'
# parse x
Improt json
y = json.loads(x)

Make template
f = open("file.html", "r")
template=f.read()
f.close()

myobj = {"user":"user","password":"password","temmplate_name":"template", "replace":"!","type":"!" ,'template': template}

x = requests.post(path, data = myobj)
val = x.content.decode('utf-8')

outputval=json.loads(val)

#make command
x = requests.get("Command")
#x = requests.get("http://localhost:8000/doit?key=Keyid&action_type=get_post”)
print(x.content.decode('utf-8') )
```

,

Commands

Note

In commands
“ is replaced with (???1???)
‘ is replaced with (???2???)
` is replaced with (???3???)
\ is replaced with (???3???)

Make User

A user password and email is added to the user database.
To perform this action user.

path/doit?user=USERNAME&action_type=adduser&password=PASSWOED&email=EMAIL

The users can be used in order to make posts private or make ledgers or do triads.

Responses:

{ "response": "ADDED_USER" }

This indicates a user has been added

{ "response": "USER_TAKEN" }

This indicates the user name is already in use

MAKE POST

Making a post is done by passing a user and password to a post. You can also do it without passing a user name or password.

***Public post Command**:*

A post that anyone can view. with the right post key

PATH/doit?user=USER&action_type=add_post&password=PASS&text=TEXT_FILD&body=BODY&photo=PHOTO&catagoy=CAT

It makes a post where the information created can be used. The text field can be modified.

See Change Post

Private post Command:

A post that can only be used with a cashed key from a ledge along with the post id

PATH/doit?user=USER&action_type=add_post&password=PASS&text=TEXT_FILD&body=BODY&photo=PHOTO&catagoy=CAT&catagoy_2=Ledgure_Name

For more information about the post key and ledger name, see the rm key and make the, ledger.

Responses:

{ "id": "POST_ID" }

The post ID where the post will be

{ "id": "NA USER_NAME to False" }

Indicates wrong user name or password

GET POST

Gets information form a post given a post id see make post

Get Private post Command:

path/doit?key=POST_ID&action_type=get_post&usekkey=USEKKEY

(For information about POST_ID, see Make POST for information about USEKKEY see RMkey)

Gets a post that is private(See Make post). Inledgero sees this post; you must use a solution key (see RM post).

If the key is wrong or there is no key you

Get Public post Command:

path/doit?key=POST_ID&action_type=get_post

The use key is obtained, by cashing a ledger with the same ledger name s the key you cash a post

Responses:

{ "id": "post_is_private+private_post_name", "username": "NA", "text": "NA", "body": "NA", "tital": "NA", "time": "NA", "photo": "NA", "iframe": "NA", "catagoy": "NA", "catagoy_2": "NA" }

Indicates that the post is private can that you need a usekkey from the ledger in order to access the post

{ "id": "post_is_NA", "username": "NA", "text": "NA", "body": "NA", "tital": "NA", "time": "NA", "photo": "NA", "iframe": "NA", "catagoy": "NA", "catagoy_2": "NA" }

This indicates the post ID does not exist.

{ "id": "POST_ID", "username": "u1", "text": "text", "body": "body", "tital": "", "time": "2021-10-09 00:43:40", "photo": "photo", "iframe": "", "catagoy": "cat", "catagoy_2": "" }

Information about the post

Change Post

Change post changes the text field of the post.

Command:

path/doit?user=&action_type=change_post&password=&key=POST_ID&text=newtext

This will change the post of a item

Responses:

{ "output": "text_updated" },

On success, it will say

{ "output": "NO_user_password" }

No user

{ "output": "NO_Post_Found" }

No Post Found

Add a Ledger

Adds a ledger to database witch can be used to generate keys

Command:

path/doit?user=USER_NAME&action_type=add_ledgure&password=top&email=EMAIL&hashword=hashword&Ledgure=Ledgure_Name

Adds a ledger that can be used to create keys that can be cashed

Responses:

{ "output": "added USER_NAME_LEDGURENAME" }

The ledger has ben added

{ "output": "taken" }

The ledger has already been taken

{ "output": "Wrong_User_Name" }

Wrong user name or password

Note that when you make a key, you must use the hashword from add ledger,

Add a Key

Adds a key to a leddgure

Command:

path/doit?ledgure=Ledgurename&action_type=add_key&password=hashword&email=Email&message=message&keymessage=keyword&keyfroward=keyforward

Note that the hashword

Responses:

{ "post_id": "KEY_ID", "solution": "SOLUTION", "email": "Email", "message": "keyforward", "ledgure_ownder_email": "owner_email", "land_url":”URL link to a cashable page", "ledgure": "Ledgurename", "path": "Path_to_websight" }

The keyname will be returned with the hashword, and an email can be sent form to ledger owner informing them.

Add change key.

Creates a new key and cashes an old one given a key name and correct solution.

Command:

path/doit?name=NAME&action_type=change_key&key=KEY_NAME&newkey=NEWKEYNAME

Responses:

{ "output": "KEY_ID" }

This key-id can is where the new key is stored.

{ "output": "NO_name_or_key_taken" }

This indicates that the key was not available or was taken.

Check Key.

Allows you to get public information about a key given a key-id

Command:

path/doit?action_type=check_key&name=KEYID

Responses:

{ "output": "KEYID", "hash": "HASH", "ledgure": "LEDURENAME", "solution": "key" }

This indicates the key has not been used.

In order to cash the key, you must have the solution to the hash, that is to say, a set of characters that when SHA256(KEY)=HASH.

Make Trade

It makes a trade so that currency can be exchanged between users.

Command:

path/doit?user=T1&action_type=maketraid&password=pass&request_type=Traid_Type&send_type=Traid_Type&request_amound=request_amount&send_amount=send_amount

Responses:

{ "response": "Wrong_Username", "amnountleft": "NA" }

Indicatats wrong user_name

{ "response": "TRAID_ID", "amnountleft": "curancy_left" }

Indicatats successful traid

{ "response": "No_Funds", "amnountleft": "NA" }

Indicates insipient funds in the account to make a trade

Print Trade

Prints information about a traid given a traid ID

Command:

path/doit?traid_id=TRAID_ID&action_type=traid

Responses:

{ "traid_id": "TRAID_ID", "traid_mony_type": "money2", "traid_request_type": "money1", "traid_request_amount": 2.0, "traid_money_amount": 1.0, "user": "USER", "buyer": "BUYER" }

{ "traid_id": "NO_traid_id", "traid_mony_type": "NA", "traid_request_type": "NA", "traid_request_amount": "NA", "traid_money_amount": "NA", "user": "NA",
Anyone can view "NA" }

Compleat Trade

Allows you to complete a triad and make users exchange the amount of money. You can find this information in the print triad

Command:

path/doit?user=T1&action_type=fintraid&password=pass&traid_id=TRAID_ID

Responses:

{ "response": "TRAID_ID" }

Indicates traid was made

{ "response": "No_Traid" }

Indicates no traid was made

Print User

Allows you to see the amount of money in a users account

Command:

path/doit?user=T1&action_type=Uprint

Responses:

{ "out": [["USER_money1", "1000.0"], ["USER_money2", "998.0"]
,[User_Othercurancy,”amount”]] }

Add Barter Currency

Command:

path/doit?user=T1&password=pass&action_type=add_C&crypto_name=b62a9b41cf2d30f009ae3bc48fc673a9d828ae30b98fe37a11d9196645d63f8c&crypto_key=db1fc35a2d1c135fda76b17e8fae63ef47af020166597daf601b80127534e849&crypto_path=http://localhost:8000/doit&L_name=T1_led

Responses:

{ "response": "NO_key" }

No key returned

{ "response": "Amount" }

The amount you got back

Get Key Back

Returns key information to user if they have the funds

Command:

{ "response": "key=Key name=Name entery_name=T1_led
path=http://localhost:8000/doit" }

Responses:

{ "response": "no_funds for T1_http://localhost:8000/doitT1_led" }

RM key

Removes or sends a key used for all sorts of purposes.

Command:

path/doit?action_type=rm_key&key=solution&name=keyname&message=message

Responses:

{ "key_message": "forward", "email": "", "forward": "T1_led", "post_id": "Usekkey", "ledgure": "legurename", "mesage_to_send": "" }

Indicates the key was cashed use key used for access to gated posts

{ "key_message": "NA", "email": "NA", "forward": "NA", "post_id": "NO_Key" }

This indicates the key was not cashed.

Template Information.

Templates are uploaded with a post command.does

```
        "User":"User",
        "Password":"Password",
        "temmplate_name":"templateName”,
        "Replace":"repalce_value",
        "type":"any"
        ,'template': example
}
```

When the page is uploaded

“' with (!A???'+replace+'???A!)
‘ with (!B???'+replace+'???B!)
` with (!C???'+replace+'???C!)
\ with (!D???'+replace+'???D!)

Where replace is your replacement value, this allows you to create templates from other templates.

This will make a page template the template name will be
Make page
path/doit?action_type=makepage&usertemplate_name=USERNAME_TemplateName&var1=Var_1&rep=Replace_Value&setion=POST_ID&setion2=POST_ID

When the templates are made
The above values are replaced and in addition.

(!Q???'+rep+'???Q!) is replace script
(!0???'+rep+'???0!) is replace Var_1 form the get var1
(!W???'+rep+'???W!) is replace &
(!S???'+rep+'???S!) is replaced with POST_ID form setion1 from the get path
(!Z???'+rep+'???Z!) is replaced with POST_ID form setion2 from the get path
(!P???'+rep+'???P!) is replaced with path to the HyperWiki form path
(!T???'+rep+'???T!) is replaced with usertemplate_name
(!L???'+rep+'???L!) is replaced with +